BACHELOR THESIS

Electrical Engineering, Electronics

College of Engineering and Architecture of Fribourg (EIA-FR)
Member of University of Applied Science of Western Switzerland

Audio Extraction from Low-Resolution 2D IRENE Images

(Acronym: AXLR)

Proposed by
Carl H. Haber
Lawrence Berkeley National Laboratory, California, United States

Author:
Yann KURZO (E3L)
yann.kurzo@edu.hefr.ch

Supervisor:
Carl H. HABER, BERKELEY LABS
chhaber@lbl.gov

Advisers:
Eric FRAGNIERE, EIA-FR
eric.fragniere@hefr.ch
Ottar JOHNSEN, EIA-FR
ottar.johnsen@hefr.ch

Expert:
Marc DE HUU, METAS
marc.dehuu@metas.ch

LBNL, August 1st, 2014

Abstract

For many years, a team supervised by Carl Haber has been studying the reconstruction of old audio recordings at the Lawrence Berkeley National Laboratory. Those mechanical audio records are usually very fragile and cannot be read conventionally. To avoid breaking any samples, the records are digitalized in 2D images and then analyzed by software.

Different methods for recovering the audio already exist and work more or less well depending on the kind of samples. The goal of this project was to develop a new method for extracting the sound from the image to try to improve the sound quality. This new method is based on the optical flow theory, part of image processing algorithms. The optical flow is usually used to find the velocity of objects (or pixels) between two video frames. In our case, it was used to recover the groove velocity on the samples.

A lot of tests were executed with different parameters on different kinds of samples, including side-lightening images, aluminum discs, and Memovox discs. The results obtained with usual discs (Shellac discs) had the same noise level with the optical flow than with the conventional algorithm. The quality of the sound was slightly improved with aluminum discs, but another acquisition system based on 3D acquisition was still better, although it need more process time.

A very interesting result was found with the Memovox discs. Those special samples were mostly recorded during the WWII and have been very difficult to analyze until now due to their geometry. However their special shape is handled perfectly by the optical flow algorithm. A good audio quality can now be recovered from those unusual recordings.

Keywords: Audio reconstruction, Image processing, Optical flow, Shellac discs, Aluminum discs, Memovox discs

Acknowledgments

I would like to thank Carl Haber for his help and critics during this project. He gave me good advices along this project and feedback when writing the report. I also want to give thanks to Earl Cornell for his help when dealing with the programming and algorithmic parts. He gave me ideas along the project on how to improve my work. He also helped me a lot for finding new audio samples to analyze.

I would like to thank Eric Fragnière, Ottar Johnsen, and Marc De Huu for their supervision. Their help was more than valuable, mainly when dealing with algorithms.

Finally I would like to thank again Carl Haber and the Lawrence Berkeley National Laboratory for offering me the opportunity of doing my bachelor thesis abroad. I also thank the College of Engineering and Architecture of Fribourg for the financial supports.

Table of contents

List of Abbreviations

Emgu CV	: Library used for the image processing (“Emgu is the Most General Unifier”)
ETH	: Eidgenössische Technische Hochschule Zürich
GB	: Giga-Byte
GUI	: Graphical User Interface
IRENE	: 2D image processing software (“Image, Reconstruct, Erase Noise, Etc”)
LBNL	: Lawrence Berkeley National Laboratory
PRISM	: 3D image processing software
RAM	: Random Access Memory
RENE	: 2D acquisition system (“Reconstruct, Erase Noise, Etc”)
WAV	: Waveform Audio File Format, standard used to store audio data stream
WWII	: World War 2

Chapter 1: Introduction

1.1 Context

For many years, a team supervised by Carl Haber has been studying the reconstruction of old audio recordings at the Lawrence Berkeley National Laboratory. Those mechanical audio records are usually very fragile and cannot be read conventionally. Some of them were even recorded before the human kind was able to replay the sound, which means that the reading machines had not been invented yet.

In order to preserve those records in good shape, methods were developed by the team to read these recordings directly without touching them. The idea was to acquire a high resolution digital image of the recordings (discs for example) and do some image processing to recover the audio signal. This method of reading recordings without contact has been ameliorated over the years, but the basic idea remains the same.

Currently the researches have been centered on two different ways of doing it: the first one is called IRENE and is based on the acquisition of 2D images. This one is mainly used for discs. The other method is based on an acquisition with a 3D probe and is usually used with cylinder media. [1]

Over the past eight years, many students from the College of Engineering and Architecture, Part of the University of Applied Sciences Western Switzerland, have been helping Carl Haber and his team in developing new pieces of software, trying then to improve the quality of the reconstructed audio signals.

1.2 Project objectives

The objective of this project is to enhance the quality of the audio signals which can be read from the images. Different algorithms are already working, but they may not result in the best sound quality. The goal of this project is to explore alternative methods to measure the groove trajectory. This includes velocity and optical flow methods and side-lightening. The objective of studying those different methods is to optimize the signal-to-noise ratio.

The new groove detection methods will be based on the optical flow theory. This method would use the groove as a succession of frames, like in a video. Getting the velocity of each pixel will result in a velocity flow. Applying the right averaging function to this flow would give the real groove velocity.

With this method, it could also be possible to analyze images obtained with side-lightening. These images theoretically contain more information, but they cannot be handled by the image processing software yet.

The modifications of the existing program for handling new analysis methods must be done with a good structure. The programing language is C# and the modifications should be object oriented. The old methods should be adapted to the new structure too.

Once the algorithm is implemented, tests to compare the results should be done. A certain amount of test samples will be selected. Silent samples can be used to determine the noise level. Different kinds of music samples will be tested to find out some differences. It is also possible to test the new algorithms on other unusual samples.

A synthesis of the new methods and parameters explaining the pros and cons will finally be written. It should summarize which methods give the best results under which conditions.

1.3 Planning

The planning of this project can be found at the end of the Appendix section.

1.4 Organization of the report

This report is divided in three main different parts. The first one will talk about the existing methods of non-invasive recordings reading. It will give a small summary of the history of audio recordings and then explain in detail the audio reconstruction processing on the 2D system which this project will be focused on. There will also be a small introduction about the different tools used along this project.

The second part is about the research itself and the different results. It will present the new method of audio reading and the theory behind it. Different tests with different samples will be exposed. Then the last part will talk about the software modifications done to the RENE program to improve its robustness. Finally the conclusion of the project and a personal overview will be written at the end.

1.5 Folder organization

The folder organization works as follows:

• Publications	
○ Admin	Administrative documents
○ Presentations	Presentations done during the project
○ Report	Final version of the report and flyers
○ Weekly report	Weekly reports and logbook
• SW_version_finale	
○ Doc	Documentation
○ Code	Final version of the modified parts of the code
• System_doc_externe	External documentation
• Test samples	Audio samples

This last folder is explained in detail in Chapter 6.1.5.

Chapter 2: Concept

2.1 History

Over the last 150 years, many different methods of recording sound have been invented. As nowadays media are usually digitalized, the earliest known methods of recording were called acoustical recordings. The first machine ever able to record a sound was invented by Frenchman Edouard-Léon Scott de Martinville in 1857. At that time, it was not possible to play back those recordings; they were basically created for studying the visual shape of the sound waves.

The first device able to both record and reproduce sounds is known as the phonograph and was developed by Thomas Edison about 20 years later. The sound itself was recorded on a cylinder covered by a layer of special material, such as tinfoil or wax. By mechanical means, a stylus could write the groove on it. In order to play it back, a needle traces the groove and its movement is then amplified, resulting in vibrations. The disadvantage of those cylindrical records is that they were not easy to reproduce for mass production.

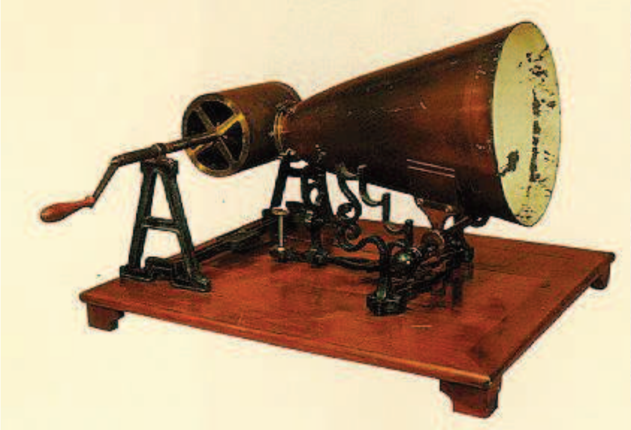


Figure 2-1: Phonautograph from 1859 [2]

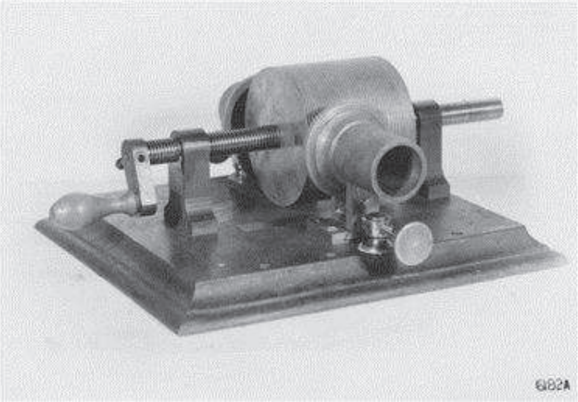


Figure 2-2: Tinfoil phonograph invented by Thomas Edison [3]

The invention of the gramophone solved that problem. Instead of etching the groove on a cylinder, they started to use discs. Those new discs were easily reproducible and gave the same sound quality than cylinders.

After the time of acoustical recordings, the development of electrical recording considerably increased the sound quality. The use of microphones and vacuum tubes for amplification gave better volume difference and frequency response to the discs.

The next big step in sound recordings was brought by the invention of magnetic recordings, such as magnetic tapes and audio cassette. Those more recent methods will not be studied during this project and the tests will only be done on disc records. [4] [5]

2.2 Groove modulation

2.2.1 Vertical modulation

On early recordings with the cylinders, the sound was recorded on the support by modulating the groove vertically. The audio signal was written by varying the depth of the stylus on the cylinder. In Figure 2-3, we can see that the width of the groove changes with the time, which corresponds to changing how deep the stylus goes.

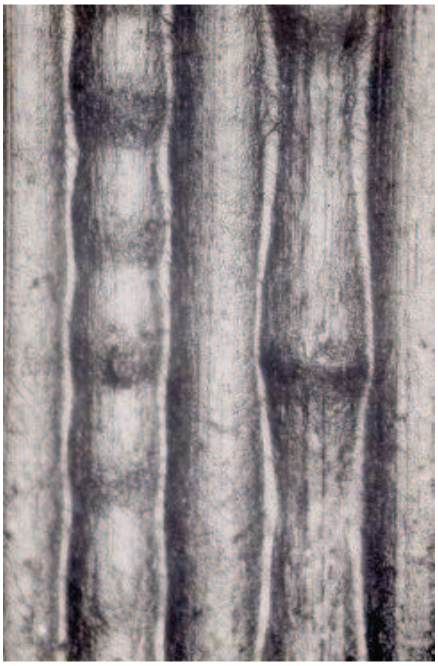


Figure 2-3: Image of a vertically modulated groove on an Edison cylinder recording [6]

2.2.2 Horizontal modulation

With the advent of the gramophone and the disc records instead of the cylinders, the vertical modulated groove was replaced by an horizontal modulated groove. This means that the depth of the groove remains constant, but its position changes.

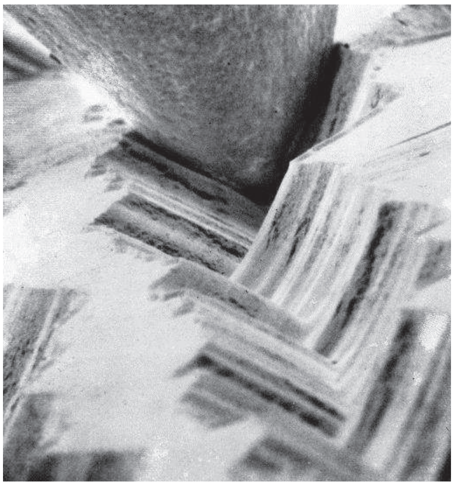


Figure 2-4: Image of a horizontally modulated groove [7]

2.3 Sort of records

In the table below, a summary of the different types of recording materials is given. It provides a small description for each of them and at what time period it was used. It also makes the difference between direct records and stamped records. The direct records were recorded in real-time. There are usually no copies of those discs so there are very precious from an historical perspective. The stamped records are initially molded on a master disc. Then a lot of copies were made from that master. This is the kind of discs which can be easily found on the market. [4] [8] [9] [10]

The groove dimensions values correspond to the schematic shown in Figure 2-5.

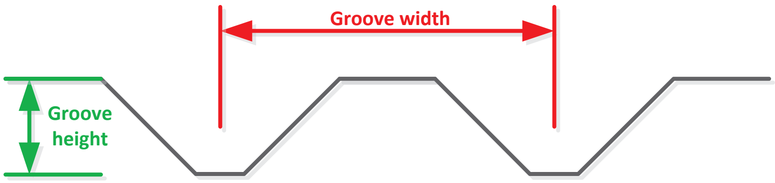


Figure 2-5: Groove dimensions

Type of materials	Description	Dates	Record type	Groove dimensions
Wax cylinders	Vertically modulated, they were used for field home recordings. They had the higher audio quality until around 1910. On the other hand, they were not easy to manipulate for mass-production.	1886-1929	Direct	Width: 125-250µm Height: 25 µm
Plastic cylinders	Used for mass-production, they were claim to be very durable	1905-1929	Stamped	As wax cylinders
Rubber	Earliest disc records	1889-1894	Direct	-
Shellac	Shellac discs were commonly used until the mid-20th century. They were weak and brittle, so they must be handled carefully. The disc tended to get bigger as the number of groove was limited to 40 grooves per cm.	1895-1950	Stamped	Width: 300-400µm Height: 75 µm
Aluminum	Aluminum discs were usually used to record radio broadcast or archives. The high frequencies were heavily attenuated, but they are not fragile at all. Most of them were destroyed during WWII, as the aluminum became an important resource for military uses.	1920-1940	Direct	Width: 300-400µm Height: 5 µm
Lacquer	(or acetate disc) Lacquer discs were rarely made for sale to the general public. They suffer more quickly than the vinyl discs. Their price can get very high at auction because of their rarity.	1930-1950	Direct	Width: 300-400µm Height: 75 µm
Vinyl	Vinyl records are soft, so that they can easily get scratches. On the other hand, they do not break easily. It can get a static charge and dust is not easily removable from it. On the other hand, they can have up to 100 grooves per cm.	1930-Today	Stamped	Width: 100-200µm Height: 25 µm

Chapter 3: Existing system

3.1 Overview

The existing system for sound reconstruction is divided into two main branches: the first one uses 2D images of the discs and the second one analyses 3D captures of the discs.

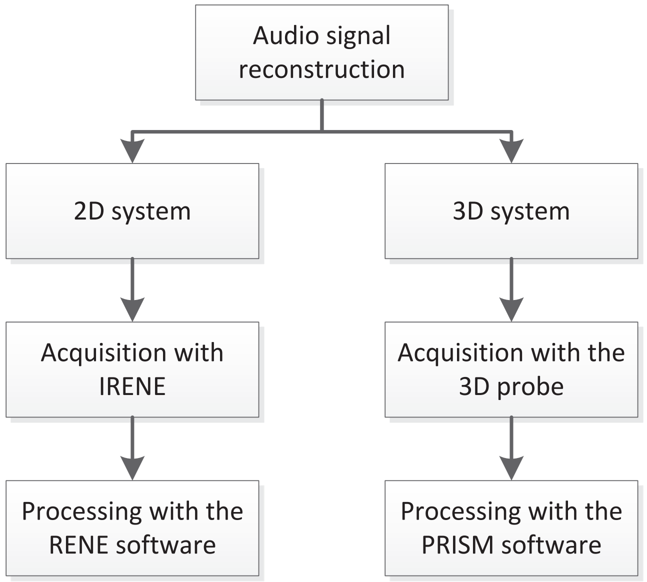

Audio signal reconstruction
2D system
3D system
Acquisition with IRENE
Acquisition with the 3D probe
Processing with the RENE software
Processing with the PRISM software

Figure 3-1: Existing system overview

Both systems get information from the discs by optical means. IRENE shoots high resolution images of the disc surface. The resulting images give the intensity in grayscale for each point. Alternatively, the 3D probe gives back the height of the different points on the disc. This allows the reconstruction of a 3D profile of the groove.

The processing done by the software (either RENE or PRISM) after the acquisition is finally quite similar. The program tries to find the edges of the groove on the image, and then, after doing some image and sound processing, it gets back the audio signal from the image.

The main differences between both systems lays in the acquisition time: IRENE produces a full image of the disc about 13 times faster than the 3D probe.

During this project, all the tests were done with the 2D imaging system. In the following chapters, explanation about this system will be given in detail. On the other hand, the 3D system will not be discussed anymore.

3.2 Image acquisition

3.2.1 Perpendicular lightening

The shape of the groove looks like the one in Figure 3-2. The top and bottom of the groove can almost be considered as flat. The side of the groove has an angle of approximately 45°.

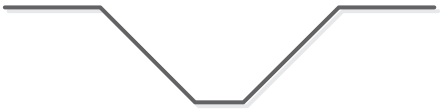


Figure 3-2: Basic shape of groove

As the discs are usually black or very dark, it is not easy to get a good contrast on the picture. The actual system uses optical fibers to bring light the disc. The beam is oriented perpendicularly to the disc as seen in Figure 3-3.

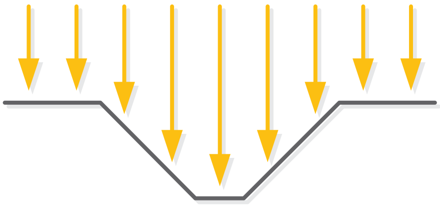


Figure 3-3: Incoming light

When the light hits the disc surface, all the beams are reflected. If the surface is flat, then the beam is reflected into the detector which is situated perpendicularly to the disc. The beams hitting the slope part of the surface will not be reflected into the detector direction, as showed in Figure 3-4.

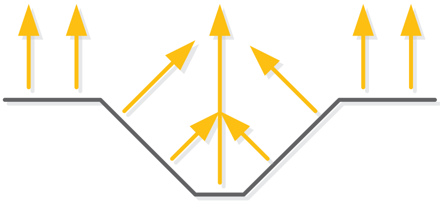


Figure 3-4: Reflected light

Such a system creates high contrasted images where the flat parts appear white and the slope parts appear black, as in Figure 3-5. These are pretty convenient for using edge detection on the groove.

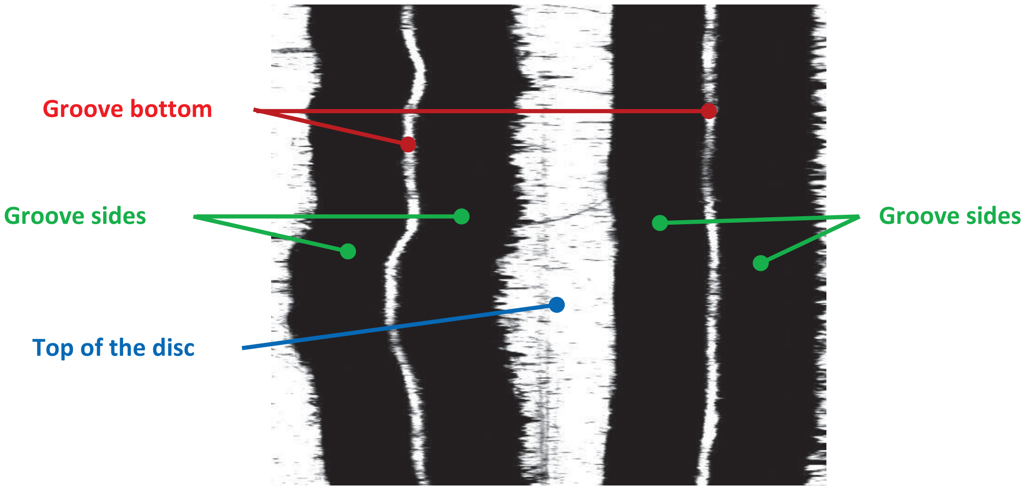


Figure 3-5: Normal lightening image from the IRENE acquisition system

3.2.2 Side-lightening images

The shape of the groove used in the previous chapter is an ideal form. The real shape of the bottom of the groove is in reality curved as shown in Figure 3-6.

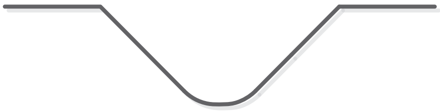


Figure 3-6: Closer to reality shape of the groove

Another type of lightening that can also be used is called side-lightening. The idea is to get more than one groove bottom by getting extra information from the sides of the groove. It is physically not possible to get the sides themselves, because the slope approaches 45°. To get a beam back to the detector, we would need to light the disc horizontally, which is not possible.

On the other hand, we can see that the bottom of the groove is not completely flat. This gives some reflections back to the detector when the light comes to the groove with an angle of 45° as seen on the next two figures.

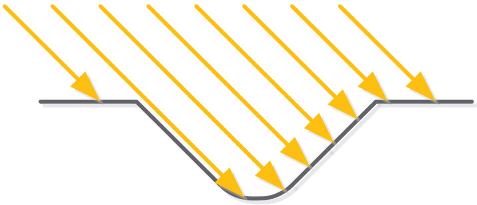


Figure 3-7: Incoming lightening

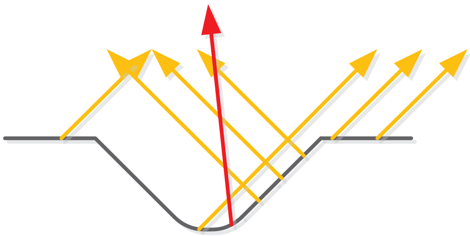


Figure 3-8: Reflected light

In that configuration, most of the beams are not reflected back to the detector. However a few beams (as the one shown in red in Figure 3-8) can be reflected in the right direction due to the fact that the groove bottom is not completely flat. When lightening the disc with two light sources coming from both left and right direction, it is then possible to get two grooves during the acquisition as it can be seen in Figure 3-9.

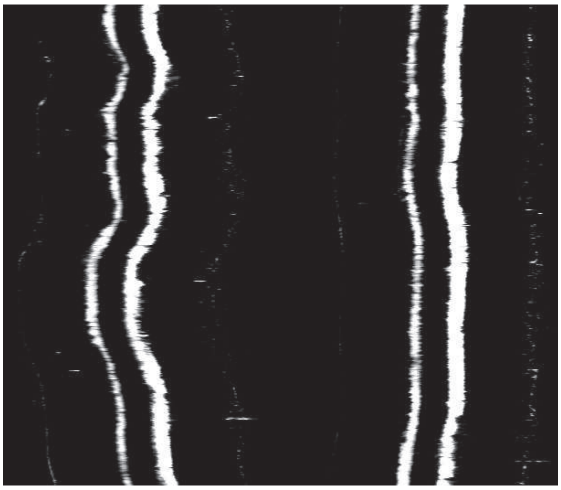


Figure 3-9: Side-lightening image from the IRENE acquisition system

Changing the angle of the lighting would get different parts of the sides of the discs. It is also possible to acquire an image with the two side grooves plus the bottom groove by using 3 different light sources together.

3.2.3 Camera specifications

To obtain high resolution images, it is not possible to take a simple photo of the entire disc. The camera sensor used by the actual system is 4096 pixels wide, but it only has a height of 1 pixel. It can only capture a line of 3.072mm at a time. In order to get the complete image of the disc, we need to take a lot of snapshots. To do that, the disc is placed on a turntable. When the disc rotates, 80 000 lines are taken at regular intervals for one resolution. The whole acquisition system, which includes motors and sensors, is controlled by software in a LabVIEW environment.

In Figure 3-10, we can see a schematic of the acquisition. The red line corresponds to the actual position of the camera. The blue ring corresponds to one revolution. The disc rotates on the turntable, so that the camera moves along the blue ring, which results in a ring shaped image.

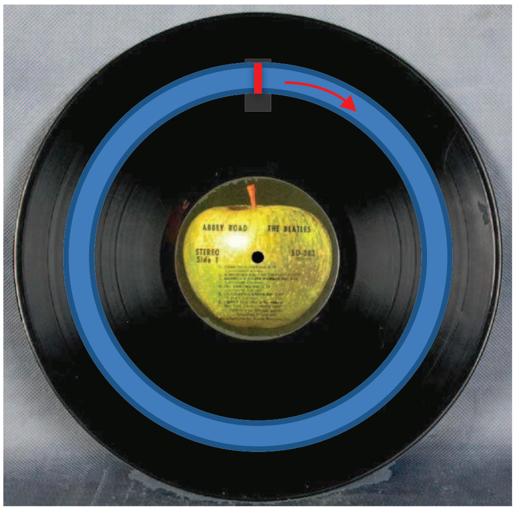


Figure 3-10: Schematic of the acquisition

3.2.4 Sampling frequency

It is important to verify that the sampling frequency of the image acquisition is sufficient for audio processing. The human ear can hear frequencies up to 20kHz. To satisfy the Nyquist-Shannon sampling theorem, the sampling frequency must be at least twice bigger, which means at least 40kHz. The sampling frequency varies with the number of samples and the revolution time, as shown in equation (3-1).

$$\mathrm{f_s} = \frac{n_{samples}}{t_{revolution}} \tag{3-1}$$

For a 78 rpm disc, the sampling frequency is given by:

$$\mathrm{f_s} = \frac{n_{samples}}{t_{revolution}} = \frac{n_{samples}}{\frac{1}{\omega_{revolution}}} = n_{samples} \cdot \omega_{revolution} = 80000 \cdot \frac{78}{60} = 104kHz \tag{3-2}$$

The sampling rate is then high enough to fulfil the sampling theorem.

3.3 Image representation

3.3.1 Mapping

As explained in Chapter 3.2.3, the resulting image from the acquisition is a ring. In order to be able to do some image processing, we need to map this ring-shaped image into a strip. A schematic of this process is presented in Figure 3-11. Notice also that both green and red lines correspond to the same position on the disc.

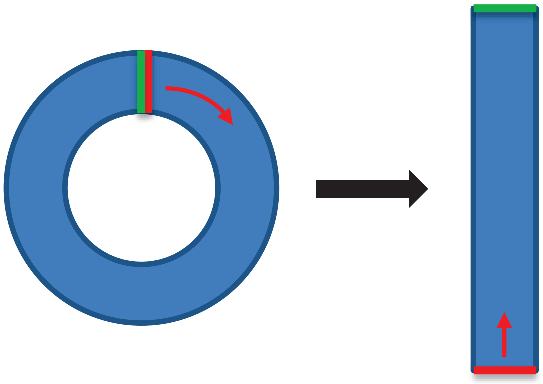


Figure 3-11: Ring to strip mapping

3.3.2 Ideal vs Real discs

When mapping the ring shaped image to the strip, the groove direction changes too. The groove has a spiral form on the disc. After mapping, the spiral becomes straight lines which are slightly inclined, as shown in white in Figure 3-12.

In practical situations, it is never that straight. The fact is that the center of the disc is never exactly in the middle of the disc. A slight off-axis of 0.5mm can already result in repeating waved lines on an entire revolution, as for example in [11].

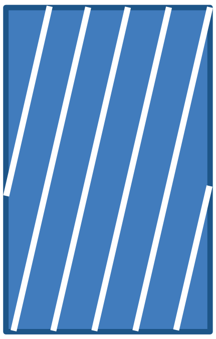


Figure 3-12: Ideal groove after mapping

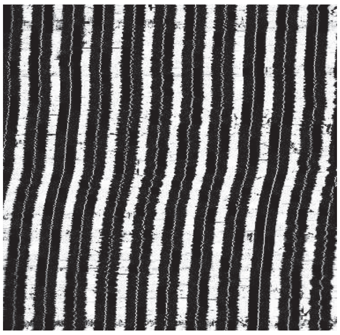


Figure 3-13: Real groove after mapping

3.3.3 Sound representation

When playing back a cylinder or a disc, the sound corresponds to the movement of the needle along the groove. When taking an image of such a disc, we can easily get the groove position, but we do not get directly the movement of the needle.

The **sound** itself does not correspond to the groove position, but it is the **velocity** of the needle. When we are analyzing a picture, we then need to calculate the velocity at which the needle would move to finally get the audio signal. This is traditionally done by taking the derivative of the groove position.

3.4 RENE program

3.4.1 Steps

The RENE program processes the images in 4 main steps as shown in Figure 3-14. Those different steps are explained in detail in the next chapters.

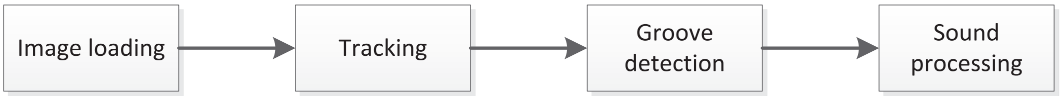


Figure 3-14: 4 main steps of the RENE software

3.4.2 Image loading

When clicking on the start button, the program asks the user to choose which image he wants to analyze. When the image is chosen, then the program converts the image to a special format described in the "FullImg" class for further processing.

As the size of the images are quite large (~320MB), they used to be divided in 8 smaller images of approximately 40MB each. When choosing the image, the program automatically checks for the 7 other images and put them back together to get the complete image. The images are stored in the bitmap format and finish by the suffix "a.bmp" where the "a" goes from "a" to "h" and corresponds to the image number.

The "tiff" format is currently under studies. This format allows the use of multiple images in one file, as if it is considered as a folder. The header of the file is also completely adjustable. Depending on the needs, different parameters could then be added to the file.

3.4.3 Tracking

Analyzing such a huge image can take a lot of time. It was decided years ago to make a first approximation of the groove position on a smaller image. For that, a binned image of the disc with a lower resolution is created from the main picture. Its size is 4 000 times 10 000 pixels. The groove is then searched to get its approximated location. Different methods of tracking exist. The "new" method is the one which usually works the best. The other methods are either older version, either used for special images such as side-lightening images. Once the tracking is done, the groove detection can take place.

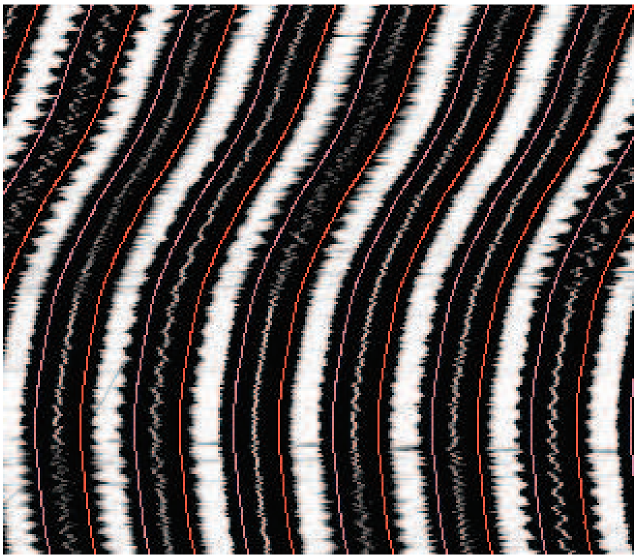


Figure 3-15: Binned image of 4000 times 10000 pixels with tracking lines in pink and red

3.4.4 Groove detection

The groove detection detects the exact position of the groove from the approximated tracking. This is the core part of the program and the quality of the audio signal at the output depends on the accuracy of the groove detection algorithm.

More details are given about the groove detection methods in Chapter 5.1.1.

3.4.5 Sound processing

The RENE software provides a few parameters for post-processing the audio signal. It implements for example a low-pass filter. It is also possible to choose the speed of the disc (rpm) and if the audio file should be mono or stereo. The audio file is then extracted in the WAV format.

Some extra sound processing is usually also applied after the end of the execution of the RENE program in a specialized sound processing software. The Sound Forge software is used for that and more information about it can be found in Chapter 4.2.

Chapter 4: Hardware and software tools

4.1 C# programming

4.1.1 Microsoft Visual Studio Express 2012

The RENE program is written in the C# language. This programming language was developed by Microsoft within the .NET framework and is now approved as a standard. It is intended to be a simple object oriented language [12].

Different IDE can be found to develop in C#, but Microsoft developed its own which is called Microsoft Visual Studio. This software is more intended for professional developers. They also created the Microsoft Visual Studio Express version which is a lightweight, easy-to-use version for students. The 2012 version is the one used during this project to modify the IRENE program.

4.1.2 Emgu CV library

Introduction

As the program does a lot of image processing, another student had the idea of using a dedicated library to avoid reinventing the wheel. This library is called Emgu CV. It provides in fact an interface between the C# language and the OpenCv library which is written in C and C++. This allows the developer to use OpenCv functions in a C# project [13].

The functions are usually well explained in the documentation, but details are sometimes missing about the algorithmic behind the function [14]. The creation of a new project which uses this library within Microsoft Visual Studio Express 2012 is not an easy task. A mini tutorial which explains the different steps can be found in the folder labeled “SW_version_finale/doc” under the name “Emgu CV project creation.pdf”.

Examples

The library provides simple functions to handle images. The type of the images can be selected when they are loaded with 2 generic parameters: the color and the depth. For example, the following code creates a grayscale image from the “image.bmp” file:

```
Image<Gray, Byte> image = new Image<Gray, Byte>("image.bmp");
```

Images are objects. The plus (+) and multiplication (*) operators are overloaded. It is then very easy to blend two images together:

```
Image<Gray, Byte> blend = 0.5 * img1 + 0.5 * img2;
```

The “Image” class also provides plenty of functions for image processing. For example, smoothing an image can be done as follows:

```
Image<Gray, Byte> smooth = image.SmoothGaussian(3);
```

Other functions and examples can be found in the documentation [14]. All those examples can of course be done with colored images too.

4.1.3 .NET Memory Profiler 4.6

The .Net Memory Profiler [15] software helps the developer in finding memory leaks and in optimizing the memory usage in .NET programs. As the C# programming language uses a garbage collector to get rid of the unused objects, it is impossible to be sure that every object is completely deleted from memory. There is also no way to know when an object is deleted. Is it for example deleted after it is used or when the program stops? This memory profiler can keep traces of every object in the program and give information about their status.

The basic idea of a memory profiler is to take more than one snapshot of the memory usage of the program at different time. Then the program analyses the differences between 2 snapshots and gives information about variables that are not regularly collected by the garbage collector. It also gives tips about where the leaks may come from, so that it becomes easier for the developer to find them.

I decided to use this program after finding out that my program was terribly slow and started to use a lot of resources (up to 4GB of RAM). More details about the memory leaks and the solutions that were found are given in Chapter 7.3.1.

4.2 Sound Forge Pro 10.0

The RENE program creates audio files from the disc images in the WAV format. These uncompressed audio files are usually opened in the Sound Forge Pro software for further analysis and filtering.

Sound Forge Pro provides plenty of tools for audio editing and analysis. It is for example very easy to apply a low-pass filter to a part of the signal. It also has a built-in spectrum analyzer which is very useful to compare two audio signals.

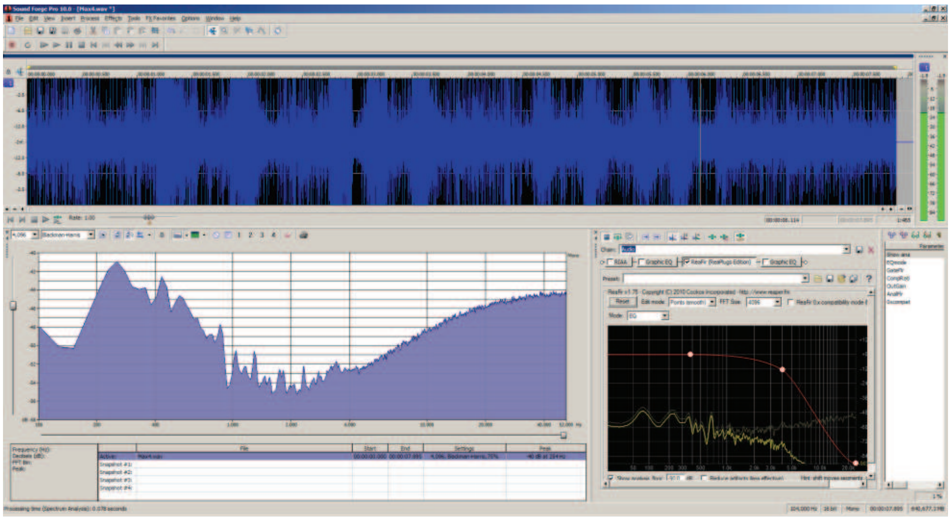


Figure 4-1: Snapshot of the Sound Forge Pro 10.0 software

Chapter 5: Groove detection

5.1 Different methods

5.1.1 Traditional methods

Most of the traditional groove detection methods are based on edge detection algorithms. By finding the position of both edges of the groove, they can then find the position of the center of the groove. As mentioned in Chapter 3.3.3, the sound corresponds to the velocity of the needle moving along the groove. To get this velocity with those groove detection methods, we need to take the derivative of the groove position.

The following method can find the center of the groove by analyzing the image and will be used to compare the sound quality obtained with the new algorithms.

Max derivative

The max derivative method has different options, and the version number 4 is the one which provides the best usable results. For each pixel, the derivative is calculated. The derivative of the groove generally has a sinusoidal shape as seen in Figure 5-1.The pixels with the maximal or minimal (highest or lowest derivative) value are defined as edges.

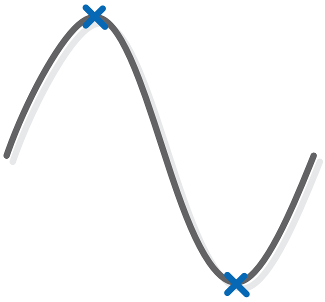


Figure 5-1: Max derivative edge detection

The algorithm tries to fit a parabola onto this derivative to get a better result. Different parameters can be taken into consideration. The default width of the groove can be defined or automatically calculated. Some smoothing can also be done to find the perfect fitting curve.

5.1.2 New method

All the traditional groove detection methods explained in the previous Chapter depend on finding the groove position. As explained in Chapter 3.3.3, the sound amplitude is represented by the groove velocity, and not its position. For all these methods, the amplitude of the sound is found by taking the derivative of the position, which works and gives good results.

The new method implemented during this project depends on directly finding the velocity of the groove from the image, without looking for the groove position. This would directly reveal the sound amplitude and may bring better results than the traditional methods.

A way to get the velocity of an object from an image is to use the optical flow theory.

5.2 Optical flow theory

5.2.1 Introduction

The goal of the optical flow theory is to find the motion of objects between two related images. The main idea is to find which pixel went where. By analyzing the intensity of each pixels between two images and knowing the time elapsed between them, it is then possible to determine where each pixel moves and at which velocity it went.

The optical flow is already used in different fields, such as motion estimation or video compression. It has been used when creating robots so that they can detect and track objects [16]. It can also be utilized to correct the camera jitter (stabilization) or to align images (mosaics) [17]. A slow-motion effect can be produced with the optical flow motion by generating intermediate frames [18]. Some sensors are also based on optical flow, such as in optical mice.

The theory is usually done when using 2D images, but it also works in 3D. In the case of our project, the theory will only be analyzed in 1D and 2D. More explanation on how the optical flow is used to detect the groove velocity is given in Section 5.3.

All the optical flow cases considered during this project were applied to grayscale images. However, it could also be applied to colored images.

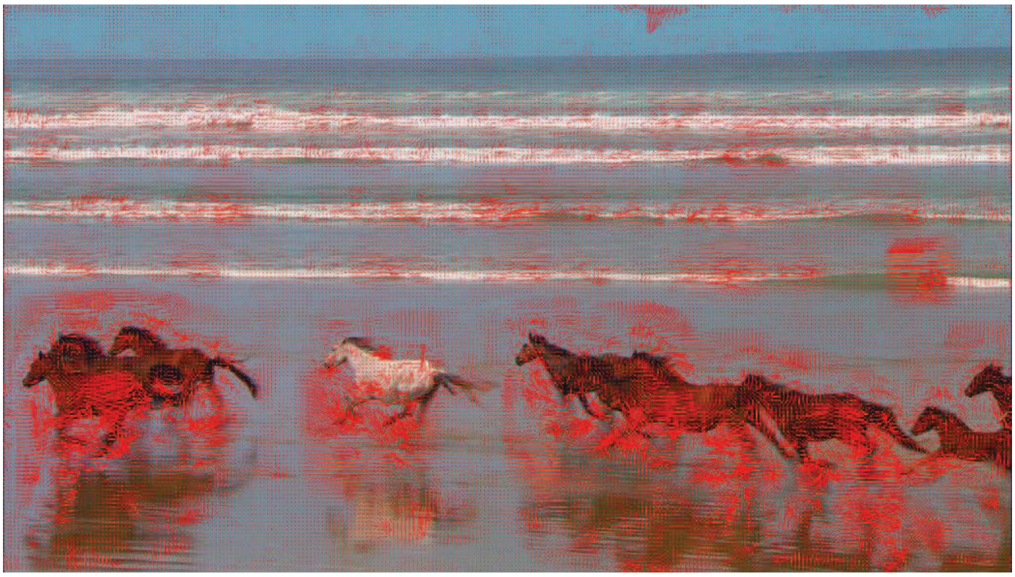


Figure 5-2: Optical flow applied to running horses; the longer the red lines are, the faster a pixel moves [19]

Most of the theory is either taken from courses ([17], [18]), or from the optical flow tutorial [20].

5.2.2 Constraint

To start with an equation, we need to consider that the brightness does not change from one image to the other. The brightness of the image is written *I* and depends on the position of the pixel:

$$I(x,y) \tag{5-1}$$

By considering the brightness as a constant, the fact that the different pixels move (**small and local motion**) from one image to the next one can be written as follows (t1 and t2 are the timestamp of each image):

$$I_{t1}(x,y) = I_{t2}(x+dx, y+dy) \tag{5-2}$$

Another way of writing the brightness of the images can be done by adding the time as a variable:

$$I(x(t), y(t), t) = I(x + \frac{dx}{dt}\delta t, y + \frac{dy}{dt}\delta y, t + \delta t) \tag{5-3}$$

The brightness constancy can also be written:

$$I = constant \rightarrow \frac{dI}{dt} = 0 \tag{5-4}$$

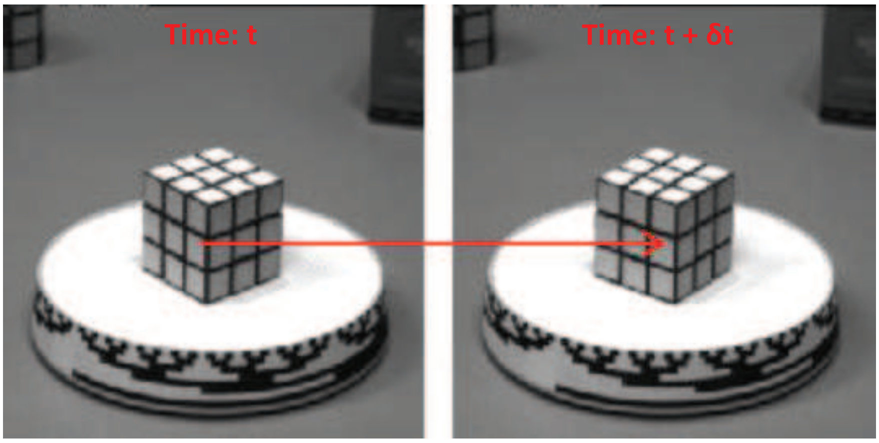


Figure 5-3: Brightness remains constant [17]

5.2.3 1D case

Let us consider first a 1D image, which just consists in a line of pixels. The brightness can then be simply written from equation (5-3):

$$I(x(t), t) = I(x + \frac{dx}{dt}\delta t, t + \delta t) \tag{5-5}$$

Applying equation (5-4) to this gives (using the chain rule for computing the derivative of the composition of more than one function):

$$\frac{dI}{dt} = 0 \rightarrow \frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial \mathrm{I}}{\partial \mathrm{t}} = 0 \tag{5-6}$$

Let us consider the following variables to simplify this equation:

$$I_x = \frac{\partial \mathrm{I}}{\partial \mathrm{x}} \qquad I_t = \frac{\partial \mathrm{I}}{\partial \mathrm{t}} \qquad v_x = \frac{\delta x}{\delta t}$$

Then the equation can be simply written as:

$$\mathrm{I_x} \cdot v_x + I_t = 0 \tag{5-7}$$

We finally can extract the velocity from this equation:

$$v_x = -\frac{I_t}{I_x} \tag{5-8}$$

To find the velocity, we can proceed by iteration. In Figure 5-4, we can see the first approximation of the horizontal velocity. The black line represents the brightness of the first image and the blue line the second image. We want to find the horizontal position of the blue point. By taking the spatial derivative I_x (brightness derivative of the first image) and temporal derivative I_t (brightness difference between both images at that point), we can get this approached green point.

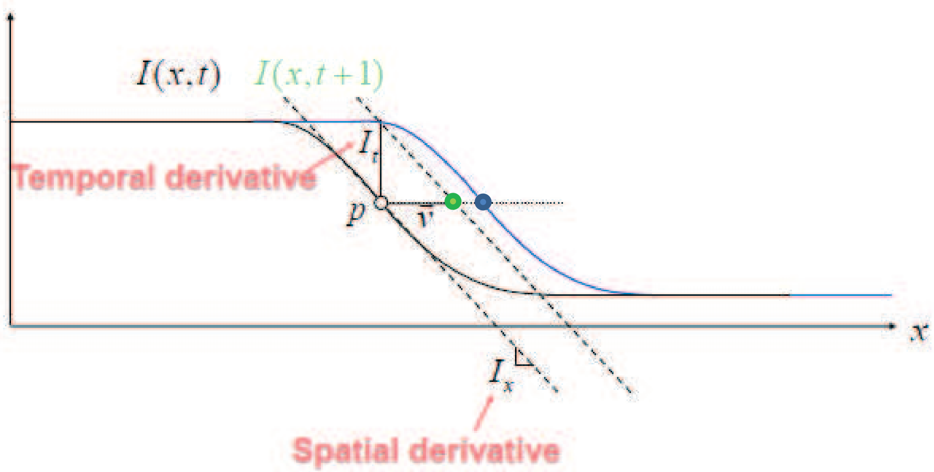


Figure 5-4: 1st approximation of the velocity [17]

As we can see in Figure 5-5, we can get closer to the real point by repeating the same process and applying the equation (5-9). I_t becomes smaller and I_x does not need to be readapted from the first approximation.

$$v_x = v_{x_{previous}} - \frac{I_t}{I_x} \tag{5-9}$$

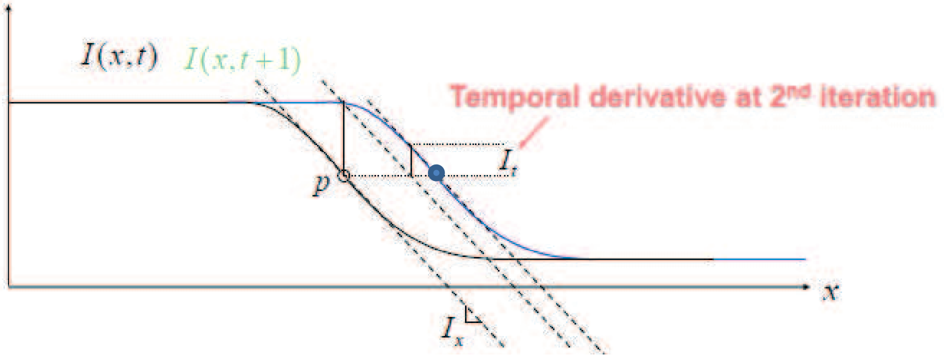


Figure 5-5: The 2nd approximation of the velocity is already right [17]

This method works when the motion is small and the brightness is mostly constant. This method usually gives correct results in about 5 iterations [17].

5.2.4 2D case

Once the 1D case is done, is seems easy to apply it to the 2D case. We take equation (5-3) and apply the derivative to get the following equation:

$$\frac{dI}{dt} = 0 \rightarrow \frac{dI}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial \mathrm{I}}{\partial \mathrm{t}} = 0 \tag{5-10}$$

By using the same variables as before and adding new ones for the y-direction:

$$I_x = \frac{\partial \mathrm{I}}{\partial \mathrm{x}} \qquad I_y = \frac{\partial \mathrm{I}}{\partial \mathrm{y}} \qquad I_t = \frac{\partial \mathrm{I}}{\partial \mathrm{t}} \qquad v_x = \frac{\delta x}{\delta t} \qquad v_y = \frac{\delta y}{\delta t}$$

We can get the equation (5-11), also named "Horn and Schunck optical flow equation". We now figure out that we have 2 unknowns for one equation. This is called the aperture problem and is discussed in next Chapter.

$$\mathrm{I_x} \cdot v_x + I_y \cdot v_y + I_t = 0 \tag{5-11}$$

5.2.5 Aperture problem

The aperture problem is the fact that depending on the motion on an image, it is not always possible to determine the exact direction of the motion without knowing more about the object in motion.

Let us look at an image through a small aperture as we can see in Figure 5-6. In each case, the motion seen into the aperture looks like if they are going in the same direction, but the movement of the lines is not. Without seeing the complete object moving, it is not possible to determine whether the object moves diagonally, vertically, or horizontally (A, B, C). In fact, the object could move in any direction if we do not know more about its shape.

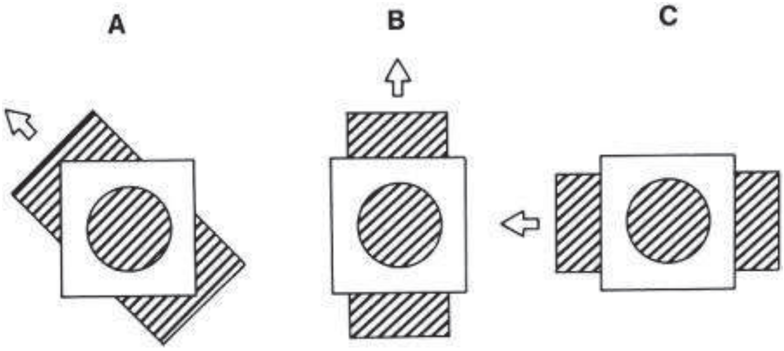


Figure 5-6: Aperture problem [21]

A good example of the aperture problem is the barber pole illusion. It consists in a diagonally striped pole which rotated around its vertical axis. Depending on the shape of the aperture, the motion seems to be done in different directions, as we can see on the two next Figures.

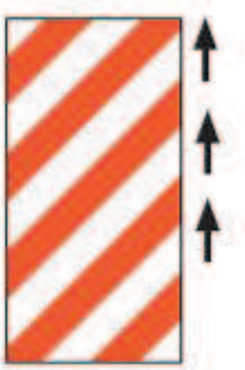


Figure 5-7: Vertical apparent motion [22]

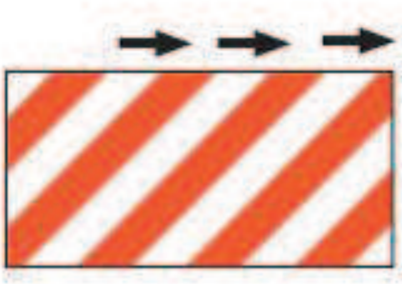


Figure 5-8: Horizontal apparent motion [22]

The aperture problem is in general solved by adding new constraints to the brightness constancy. Those new constraints give more equations, so that the flow can be found for the complete image.

5.2.6 Algorithms

Different algorithms exist to perform the optical flow. All of them give as a result the velocity (direction and intensity) of each pixel moving between the two images. We can find some examples in the list below. [16]

Horn-Schunck method

The Horn-Schunck method introduces a smoothness constraint to avoid the aperture problem. It assumes that the flow should be smooth to a certain level on the whole image. It usually yields to a dense flow vectors, but it is slightly sensitive to noise.

Lucas-Kanade method

The Lucas-Kanade method assumes that the flow is more or less constant in a small neighborhood of pixels. It then tries to solve the optical flow equations by using the least squares criterion.

Block-based methods

This method analyses the images by blocks and tries to minimize the sum of squared differences between them to find the best match.

More explanation about the implementation of these methods can be found in the course labeled "Optical Flow" from the ETH [18].

5.2.7 Optical flow with Emgu CV

The Emgu CV library implements a few methods for performing the optical flow. Those methods can be found in the "OpticalFlow" namespace and can be executed statically in the program.

The Horn-Schunck and Lucas-Kanade methods are available in the library. There is also a function using the block-based method. Some other methods also exist, but they did not give any good results.

5.2.8 Problem with the brightness constraint

For most optical flow algorithms, we consider that the global brightness between both images is constant. It may not be the case here, but it seems that the usual optical flow algorithm do not react too badly to these brightness changes. Researches have been done for applying optical flow algorithms to images on which the brightness is not constant. Those methods will not be further developed here, but papers can be found in the bibliography ([23], [24], [25]).

5.3 Application of the optical flow to the groove detection

5.3.1 Explanation

As already mentioned, we know that the sound corresponds to the velocity of the stylus moving along the groove. Applying an optical flow algorithm would directly result in obtaining the velocity of the groove instead of its position. The main question is which image should be analyzed. The classical way of using optical flow is done on two images taken at different moment in time.

In our case, we have only one huge image. The time component is already part of that image. Let us take this image magnification of the groove as an example:

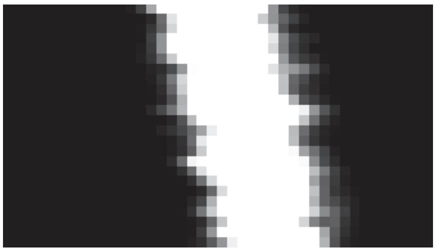
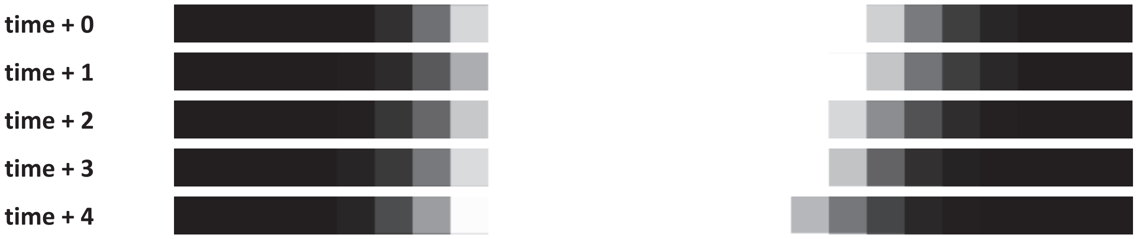


Figure 5-9: Image magnification of the groove

If we cut picture horizontally in order to obtain a lot of slices, we can say that each slice represents the position of the groove at different moment in time:



In that way, we can finally apply the optical flow algorithm between two slices. It will basically finds a velocity vector for each pixel on the image (the images must have the same size and must be correctly aligned).

5.3.2 Block diagram

In Figure 5-10, we can see the block diagram of the optical flow algorithm. It shows the different steps needed to obtain good results for detecting the groove with the optical flow method.

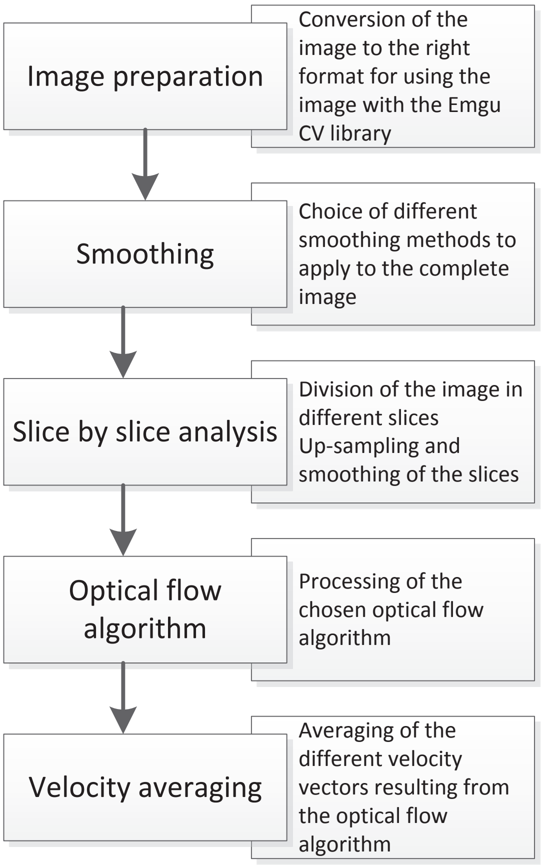


Image preparation: Conversion of the image to the right format for using the image with the Emgu CV library

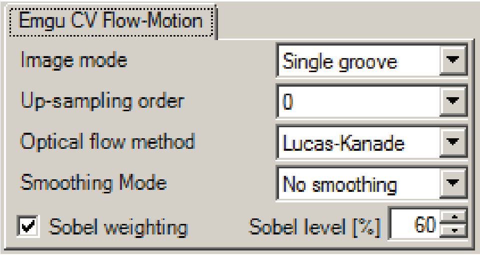
Smoothing: Choice of different smoothing methods to apply to the complete image

Slice by slice analysis: Division of the image in different slices Up-sampling and smoothing of the slices

Optical flow algorithm: Processing of the chosen optical flow algorithm

Velocity averaging: Averaging of the different velocity vectors resulting from the optical flow algorithm

Figure 5-10: Block diagram of the optical flow algorithm



Emgu CV Flow-Motion

Image mode: Single groove

Up-sampling order: 0

Optical flow method: Lucas-Kanade

Smoothing Mode: No smoothing

☑ Sobel weighting   Sobel level [%] 60

Figure 5-11: User interface for the groove detection method based on optical flow

The user interface of the optical flow groove detection method is shown in Figure 5-11. The different options as well as the functionality of each part of the block diagram will be explained in detail in the following chapters. Details about the implementation can be found in Section 7.1.

5.3.3 Image preparation

After loading the image in the RENE software, the pixels are stored in a “FullImg” class as explained in Chapter 3.4.2. For applying most of the optical flow algorithms, we need to work with the image class provided by the Emgu CV library. As it is not the goal to rewrite the whole program, we need to find a way to convert the image without modifying the part of the program which already exists.

Two ways to achieve this conversion could be done:

- Retrieving the bitmap filename of the image and directly load it in the Emgu CV image structure
- Convert the “FullImg” class directly to the Emgu CV image structure

The second way was chosen. It is probably not as fast as loading the image directly from the hard drive, but it is easier to implement with the actual structure of the RENE program. The program has different ways of loading an image (complete image or divided images for example). Therefore it is safer to convert the “FullImg”, because we know that it can handle all kinds of images.

The “ImageInterface” class located in the “utils” folder of the project was written to provide this conversion function. It is also possible to convert just a part of the image by defining a working rectangle on the full image.

5.3.4 Smoothing

After a few tests, it was observed that smoothing the complete image before applying any optical flow tends to give better results. A bad optical flow result is often due to the presence of black spots or cracks on the image. Smoothing tends to minimize their amplitude. The aftereffects of smoothing the image are that it creates a low-pass filter effect on the audio signal. As discussed at the laboratory, we want to avoid any filtering on the audio signal. The people who ask for recovering audio from discs are usually archiving the results and they prefer to get the raw audio signal and apply some filtering themselves afterwards.

When using the optical flow groove detection method, it is possible to set or disable the filters on the GUI. By default, it will be disabled.

The different used filters are presented bellow and are applied to the Figure 5-12 to see how they react.

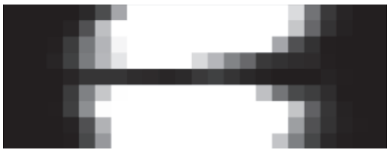


Figure 5-12: Groove with a crack in the middle

Gaussian filter

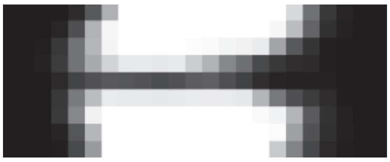
The mainly used filter is a Gaussian filter. It is only applied in the y-direction (that means the time) to suppress black spots or cracks on the image. We do not smooth in the x-direction to avoid modifying the groove edges. Its effect is non-negligible and the result usually has a better sound quality. It creates a low-pass filter on the audio signal at around 15kHz, but the lower frequencies have a better shape.

The Emgu CV library provides a function for applying the Gaussian filter to an image with different parameters. It is possible to define the width and height of the kernel. The standard deviation of the Gaussian can also specified for both axes. The Gaussian function in our case is (σ is the standard deviation):

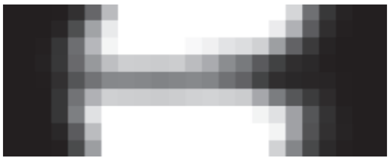
$$G(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} \tag{5-12}$$

The following examples are Gaussian filters with a different standard deviation applied to the image in Figure 5-12:

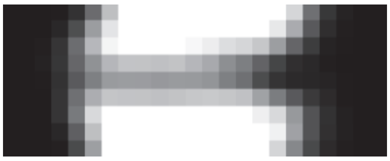
$$G(\sigma = 0.5) = \begin{bmatrix} 0.11 \\ 0.80 \\ 0.11 \end{bmatrix}$$



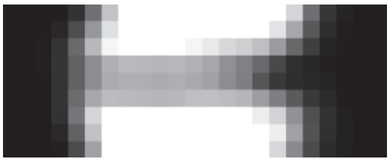
$$G(\sigma = 0.75) = \begin{bmatrix} 0.22 \\ 0.53 \\ 0.22 \end{bmatrix}$$



$$G(\sigma = 1) = \begin{bmatrix} 0.24 \\ 0.40 \\ 0.24 \end{bmatrix}$$



$$G(\sigma = 2) = \begin{bmatrix} 0.18 \\ 0.20 \\ 0.18 \end{bmatrix}$$



We see that for a standard deviation of 0.5, there is almost no effect on the image. On the other hand, a standard deviation above 1 tends to do the average between the pixels. In our case, we need to smooth to get a gray gradient so that the optical flow works better. A standard deviation of 1 will be chosen for the tests.

Partial smoothing

The bad point of smoothing the whole image is that it affects the audio signal. Another idea was to smooth only the parts of the image which have those black spots or cracks. The partial smoothing option does that by scanning the image and applying a Gaussian filter only when necessary. It also creates a low-pass filter, but it has much less effect on the audio signal than the complete smoothing.

Median filter

The median filter is a non-linear filtering technique. It does not have a specific kernel. The main idea of this kind of filter is to go through the image and replace each pixel with the median of the neighboring pixels. Let us take some examples with a 3 by 3 kernel.

Before	$\begin{bmatrix}0 & 0 & 0\\ 1 & 1 & 0\\ 1 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\\ 2 & 6 & 0\\ 2 & 2 & 1\end{bmatrix}$
After	$\begin{bmatrix}0 & 0 & 0\\ 1 & 0 & 0\\ 1 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 2 & 1\\ 0 & 1 & 0\\ 1 & 2 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 2 & 2 & 1\end{bmatrix}$

The advantage of this filter is that it completely replaces the out-of-bonds pixels. On the other hand, it replaces the bad pixels by existing pixels. The resulting image does not have a smooth gray gradient anymore, but bad pixels are repeated as we can see in Figure 5-13.

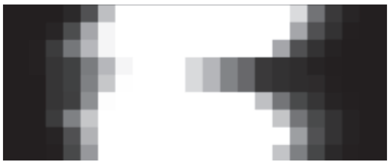


Figure 5-13: Result of the median filter (3x3)

Interpolation

The problem of smoothing the image is that it applies a low-pass filter to the audio signal. Another idea to avoid smoothing and this low-pass filter was to do an interpolation when the data returned by the optical flow algorithm do not seem good enough.

The principle is to detect big changes of velocity between two consecutive vectors. If the difference is bigger than a certain threshold (the velocity change goes to fast), then the problematic vector is corrected by doing a cubic spline interpolation. The program looks for two cases: one bad vector or two bad vectors. In both cases, we take the 2 previous and the 2 next good velocities to correct the bad ones by applying the cubic spline interpolation.

This method tries to remove cracks, but it does not work quite well. The algorithm used to detect a crack is probably not very efficient, which sometimes results in worse audio quality.

5.3.5 Slice by slice analysis

As previously mentioned, the optical flow algorithm itself will be applied slice by slice to the image in order to retrieve the velocity of the groove. For each slice, a 2 pixel height image is taken from the complete image for applying the optical flow algorithm. Some processing is done on this image before they get separated and considered as frame one and two for the optical flow algorithm.

Up-sampling

The algorithm works much better if the pixels are neither completely black nor white. In Figure 5-14, we can see the 2 pixel height image which should be analyzed. It does not have a lot of pixels which will really be taken into consideration when applying the optical flow algorithm. Gray gradients are the parts of the image which gives the better results, but there are only a few of them there.

A way to try to avoid that problem is to up-sample the image to get a better gray gradient between the white and black pixels. Up-sampling increases the number of pixels, but does not change the information contained in the image. The Emgu CV library provides a function called “pyrup” which does that for us [14]. The result of up-sampling one slice can be seen in Figure 5-15.

Figure 5-14: Raw slice

Figure 5-15: Up-sampled image

The Emgu CV documentation gives some basic information on how the function works. The method is the same as when up-sampling electrical signals. Zeroes are inserted between the rows and the columns. The resulting image is then convolved with a smoothing filter. Up-sampling does not add data to the image. However it is possible that the image is slightly distorted by the smoothing filter, because a perfect filter does not exist.

The flaw of the up-sampling is that the images become much larger and the analyzing time gets longer. The up-sampling parameter was one of the first parameter to be implemented. Due to some new parameters, it was found that changing the up-sampling order does not change the result anymore. It will most of the time be kept to 0, because it is faster.

Slice smoothing

After a few tests, it was decided to smooth the slices with a Gaussian filter (see Chapter 5.3.4 for more information about this filter). This improves the results, because the gray gradients are more regular. Furthermore it does not create a low-pass filter on the audio signal, as it is just a local smoothing.

The image is then finally divided into 2 slices in order to be used with the optical flow algorithm.

5.3.6 Optical flow algorithm

The portion of code applying the optical flow algorithm is quite easy. It is finally just a call to the Emgu CV function with the right parameters. Three algorithms have been retained and can be chosen:

- OpticalFlow.LK(): Lucas-Kanade method
- OpticalFlow.HS(): Horn-Schunck method
- OpticalFlow.BM(): Block-based method

All of them result in two images: the first one contains the x-direction velocity vectors for each pixel and the other one the y-direction. The algorithms find a vector for every pixel. Therefore we need to discard the ones which do not correspond to the groove.

To do that, it was decided to discard every pixel where the grayscale is constant. By doing that, we would only keep the pixels where the grayscale changes, which correspond to the groove edges. To determine a grayscale change, the derivative of the image is taken. The derivative of the image can be calculated with the Sobel operator from the Emgu CV library.

As the gray gradient is in the horizontal direction, the following kernel with a size of 3 is applied by calling the Sobel function:

$$G_x = \begin{bmatrix} -1 & 0 & -1 \\ -2 & 0 & -2 \\ -1 & 0 & -1 \end{bmatrix} \tag{5-13}$$

The following image shows the groove normal image and its resulting derivative:

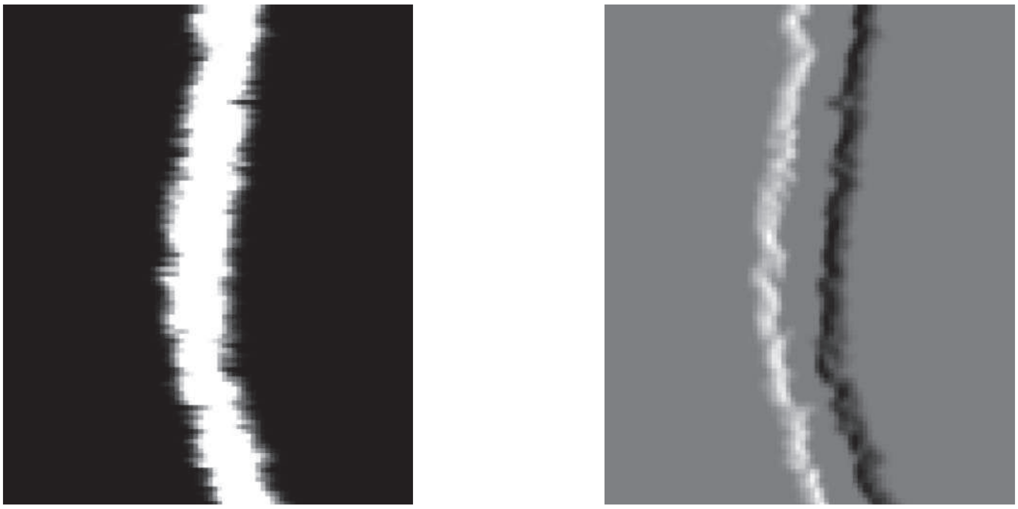


Figure 5-16: Effect of the Sobel operator on the groove

The Sobel operation results in a grayscale image (the intensity is coded between 0 and 255). A medium intensity (approximately 128) means that the derivative is null. Values higher than 128 indicate a black to white change, and lower values than 128 indicate a white to black variation. In order to be able to discard the bad vectors, the absolute value is taken and is normalized between 0 and 1 with the equation (5-14).

$$Derivative_{normalized} = \frac{|Derivative - 128|}{128} \tag{5-14}$$

The level at which a velocity vector is considered as a good vector can be adjustable in the user interface. The value is entered in percent. Each vector higher than the value is taken into account for the averaging process. The other ones are simply discarded.

The sobelized image is also used for another purpose. The different velocity vectors can be weighted by the derivative levels. This means that a vector which is found where the grayscale varies rapidly will have a higher weight when calculating the average.

5.3.7 Velocity averaging

Once we get all the velocity vectors from the optical flow algorithm, we need to average them to get the velocity of the groove. There are different ways of averaging the vectors. The averaging method is not configurable, but all the methods exist in the code and can be changed with the "#define" lines in the file labeled "VelocityVectorArray.cs".

Classical averaging

This classical way of averaging simply gets all the velocity vectors and averages them. It takes into account the weights from the sobelized image if necessary.

Sort and discard averaging

This method sorts all the velocity vectors and then only keeps the middle values. A certain number of the bigger and smaller values are simply discarded. The average is then made on the left values. This averaging method does not take into account the weights from the sobelized image in any case.

Edge averaging

In Figure 5-17, we can see in red the detected velocity vectors. We see clearly that they are two edges. Sometimes, both edges do not go exactly in the same direction. Some other times, the number of detected velocity vectors is not the same for each edge.

The idea of this method of averaging is to average the velocity vectors of each edge separately. Only then, we do the average of those averages. As the number of vectors is not always the same for each edge, it is not equivalent to the classical averaging.

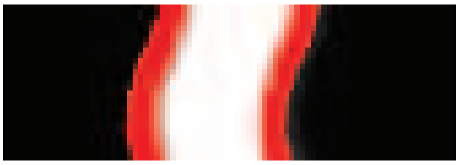


Figure 5-17: Flow on 2 edges

5.3.8 Summary of the different parameters

A summary of all the different parameters is given in the table below. The GUI option column means that the parameter can be changed on the user interface when the program is running. If it is not the case, it must be changed in the code.

Parameter	Choices	GUI option
Smoothing	Setting the smoothing method. Most of them create a low-pass filter effect on the sound. If this is not desired, the No smoothing method should be selected. • No smoothing • Partial smoothing • Gaussian filter • Median filter • Interpolation (no smoothing)	**Yes**
Up-sampling order	Setting the up-sampling order. This parameter is usually put to 0, but it may help to set it to the 2nd order with images with a single groove (no side-lightening). However the processing time increases. • 0 (no up-sampling) • 2	**Yes**
Optical flow method	Setting the optical flow algorithm. Lucas-Kanade is the one usually selected. • Lucas-Kanade • Horn-Schunck • Block-based	**Yes**
Sobel level	Select from which level a velocity vector should be kept for the average. A value of 60% is the default parameter. *Adjustable level for discarding velocity vector*	**Yes**
Sobel weighting	Using the sobelized image to weight the velocity vectors. This parameter should in general be checked. *Use of the sobelized image to weight the velocity vectors*	**Yes**
Averaging	Select the averaging method. The classical is the one that gives the better results so far. • Classical averaging • Sort and discard averaging • Edge averaging	**No**

5.4 Application to side-lightening images

5.4.1 Image acquisition

There are two types of images using side-lightening which can be utilized during this project. Depending on the way with which the images are acquired, the resulting image may be slightly different. The two ways are the following:

- Using side-lightening only: the resulting image is a 2 groove image. It is then possible to add by software the normal image with the groove bottom to this image to obtain a 3 groove image. This method has a big flaw which is explained in Chapter 5.4.2.
- The second method uses 3 different light sources. The result is that we directly obtain a 3 groove image. If the setup of the acquisition system is correct, this way normally gives better images.

Using side-lightening images basically increases the quantity of data on the image. Doing the average of more data theoretically decreases the noise level in comparison to the signal level. Those images should give better results and it is up to now only possible to analyze them with the optical flow algorithm.

It is sometimes possible that some parts of the grooves are not visible on the picture. If it is the case, it is necessary to interpolate the missing information. With side-lightening, it is possible that other grooves (bottom or sides) are visible. The interpolation would not be needed and the result would have a better quality. In Figure 5-18, we can see that the 2 grooves to the left have some cracks, but the one to the right is completely fine.

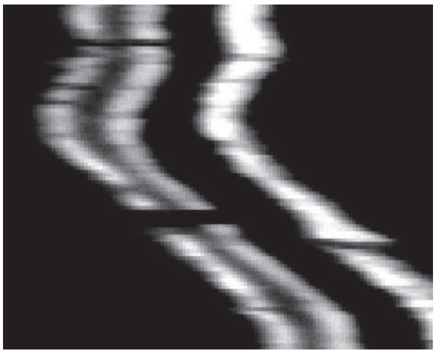


Figure 5-18: Example of a 3 groove image with cracks on some grooves

5.4.2 Superposition problem

The flaw of those 3 groove images is that some parts of the groove are sometimes superposed, which could result in wrong velocity vectors detection. When using the first way for side-lightening images acquisition, it is not certain that the lights have exactly the right angle. The resulting image can be not completely centered compared to the groove bottom. When adding them together by software, they could be superposed.

This really depends on the images. In Figure 5-19, we can see a superposition without problem (though the 3 grooves are quite close to each other). In Figure 5-20, we clearly see that the side-lightening was not centered and is completely superposed to the groove bottom.

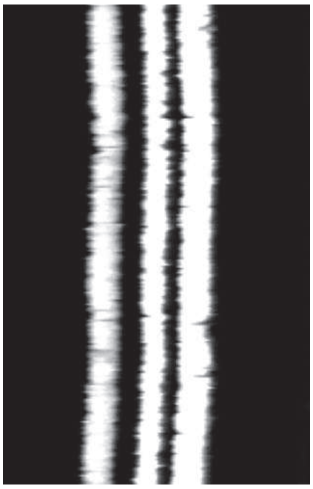


Figure 5-19: Good 3 groove image

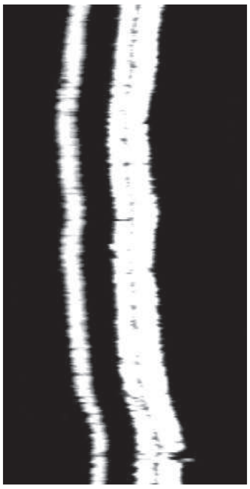


Figure 5-20: Groove bottom and side superposition

5.4.3 Optical flow algorithm

The advantage of the 3 groove image is that it has more pixels on which the optical flow algorithm can rely on. The more edges the image has the better the optical flow becomes. When using the optical flow algorithm with only one groove, the up-sampling parameter can be used to have more pixels to analyze. With side-lightening images, the up-sampling is not really needed because the number of usable pixels is high enough.

The optical flow is theoretically able to handle any amount of edges, even if they are to sharp. A grayscale gradient is still preferable. If more pixels give back a velocity vector for one slice, the averaging should minimize the noise and only get the real signal.

5.5 Adaptation to RENE software

The adaptation to the RENE software and details about the implementation are given in Section 7.1 of this report.

5.6 Summary

The optical flow method of groove detection is a complete new method that has not been utilized in that field yet. The number of parameters to adjust is quite large and optimal parameters must be found in order to get the best results from this algorithm.

The theory behind the optical flow is quite complicated and the different algorithms that can be used have not been developed in detail. However the libraries provide built-in functions which help the programming part quite a lot. The side-lightening images should theoretically give a good quality enhancement compared to the normal images with one single groove.

Chapter 6: Tests on sample recordings

6.1 Method of comparison

6.1.1 Way of testing

The tests are divided in 3 sections. The first tests were done to compare the different optical flow methods used to recover the signal. Before comparing the new methods to a traditional method, these tests were done to find out with which parameters the optical flow methods give the best results (see Section 6.3).

Then some extra tests were done to setup the other parameters. These tests are described in Section 6.4. Those parameters do not change the quality of the resulting signal that much, but they may give a little improvement. The values found during these tests will be used as a basis for all the other tests. It includes the Sobel level and weighting, the optical flow algorithm used, together with the averaging method.

After those tests, 2 optical flow methods will be selected to compare them to the best traditional method, which is the Max derivative number 4 (see Section 6.5). A summary of the results can be found in Section 6.6.

6.1.2 Sound processing

Doing some sound processing on an audio signal may increase its quality, but the goal of this project is to increase the quality of the image processing. Therefore, the only sound processing operation that may be applied to the audio signals is adjusting the volume of the samples. It is indeed important to have signals with the same amplitude in order to be able to compare them correctly.

6.1.3 Spectrum analysis

It is not easy to compare audio signals. The simplest way would be to listen to it. But this way can only give an appreciation of the quality of the signal. It is possible to say that the noise sounds louder or different when listening to 2 different samples. Furthermore not everyone's ear works the same. It is usually said that the human can hear between 20Hz and 20kHz, but this is true for young people. The ear capacity to listen to certain frequencies changes over time.

For this project, it was decided to do a spectrum analysis to compare signals to each other. On the Sound Forge software, it is possible to display the frequency response of different signals. The parameters of this FFT are fully adjustable. The settings used during this project are:

- Blackman-Harris window
- 4096 points
- The FFT overlap is set to 75%, higher values do not change the spectrum, but the FFT is then longer to process
- The length of the samples is between 3 and 8 seconds depending on the samples

The way to interpret the results is described in the next chapter.

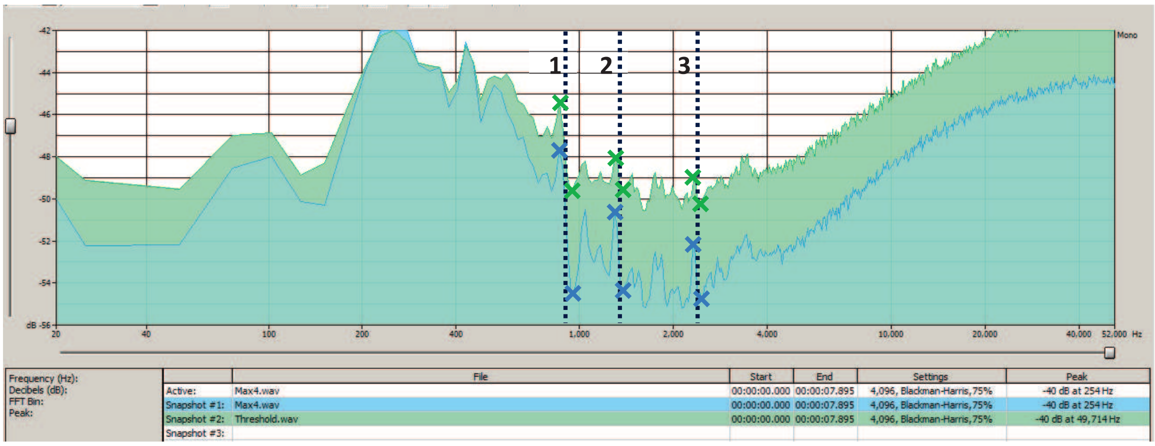
6.1.4 Interpreting the results

A first method to verify how a method reacts is by comparing silent samples and checking the level of the noise over the frequencies. This will give a basic idea of the algorithm quality, but it does not compare the noise to the signal level.

A method discussed at the lab is to compare the level difference between a peak and a valley next to it. In Figure 6-1, we can see an example between the Max derivative number 4 algorithm and the threshold algorithm. It is known that the second one gives really bad results. If we take the absolute difference for the first labeled peak, we find approximately 7dB for the blue spectrum and 4.3dB for the green one. The other values are written in the table below:

	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	7	4.3	2.24	1.64	27
Point 2	3.9	1.7	1.57	1.22	22
Point 3	2.7	1.3	1.36	1.16	15

We can then say that the blue spectrum is approximately 20% better than the other one.



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	Max4.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-40 dB at 254 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-40 dB at 254 Hz
Snapshot #2:	Threshold.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-40 dB at 49,714 Hz
Snapshot #3:					

Figure 6-1: Comparison between a good sample (in blue) and a bad sample (in green)

The difference between the peak and the valley will be called the signal-to-noise ratio (or **SNR**). In this example, the SNR for the blue curve at the point number 1 is 7dB (and 4.3dB for the green curve).

It is important to signal that the bottom of the valley does not necessary correspond to the noise itself, but it is related to the background audio. The relative difference will be the main way to compare signals in the following tests. The number of points is arbitrary, but it is usually taken where the level difference between the peak and the valley is relevant.

This way of giving results is not an absolute measure technique. It is more a tool to compare the noise level between the different samples.

6.1.5 Folder organization

The different samples that will be tested are stored in the folder labeled “Test samples” situated at the root of the project folder. This folder contains 2 sub-folders:

- “Silent samples”: Samples containing only noise (see Chapter 6.2.2)
- “Music samples”: Samples containing real music (see Chapter 6.2.3)
- “Special samples”: Not usual discs (see Chapter 6.2.4)

Each sample is stored in a different folder to avoid mixing the results. The names of the folders correspond to the names given by the research team of the lab.

For each test sample, an “Audio” folder will be created to store every audio results got from the RENE software. Because different algorithms can be used, I decided to use the following convention to name them:

- “Max4”: Used for the results found using the Max derivative number 4 algorithm

- “OF<*parameter*>”: Used for the results found using the optical flow algorithm

The parameter contains a list of the different options which can be chosen for the optical flow algorithm. It has the following format:

- *<ImageType><Order><Smoothing>*

Each parameter can have the following values:

- ImageType (Section 5.4)
  - Bottom: Only the bottom groove
  - Side: Only the side grooves (2 grooves)
  - All: Bottom plus side grooves (3 grooves)
- Order (Chapter 5.3.5)
  - Up0: No up-sampling
  - Up2: Second order up-sampling
- Smoothing (Chapter 5.3.4)
  - NoSmoothing: No smoothing
  - PSmoothing: Partial smoothing
  - GSmoothing: Gaussian filter
  - MSmoothing: Median filter
  - Inter: Interpolation

The other parameters are set to the values found in the extra test chapter (see Section 6.4 and Chapter 6.4.6 for a summary of the used parameters), because they do not change the results quality that much.

6.2 Samples overview

6.2.1 Introduction

In order to get a good idea of the quality of the new algorithm, it was decided to analyze the results with silent and music audio samples. For every test, we used both the normal image with the groove bottom only and images with the groove sides. An extra test will be made with a 3 groove image and it will be specified when it is the case.

Finally I also did some tests on some special samples with which the Max derivative number 4 algorithm does not react very well. Those samples are described in Chapter 6.2.4.

6.2.2 Silent samples

Analyzing “silent” samples can be very interesting to see how the algorithm reacts when no sound is played. The result is only noise, but it can be at different levels depending on the groove detection method.

It is not very easy to find silent samples. Most of the samples here are taken at the beginning of the songs where no music is recorded yet. Those “leading” grooves were used to put the stylus at the right position before the music started. The “Silent samples” directory contains these following shellac discs:

- DC0
- LDTruck0
- Kansas1

6.2.3 Music samples

The “Music samples” directory contains images and sound of the following discs. They were taken more or less randomly from the lab archives and are all shellac discs.

- LDTruck5
- Nobody16
- SSB25
- DC1

All of them include bottom groove images and side-lightening images. The “DC1” sample also has a 3 groove image which was taken trying to avoid the superposition effect (see Chapter 5.4.2 for more details about the superposition).

6.2.4 Special samples

After figuring out than most of the shellac discs give similar results with both the Max derivative number 4 and the optical flow methods, some other types of records were searched to try the new algorithm. Those records are special and do not give good results with the usual methods. A few examples are given here with some explanation.

Aluminum discs

Some special aluminum discs which seem to have a “W” shaped form were found in the archives. In the two figures below, we can see to the left a normal acquisition and to the right a side-lightening acquisition.

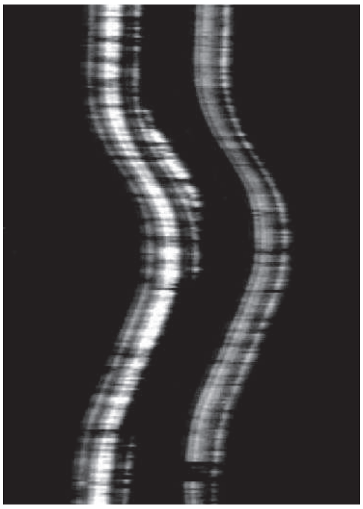


Figure 6-2: Normal acquisition of aluminum disc

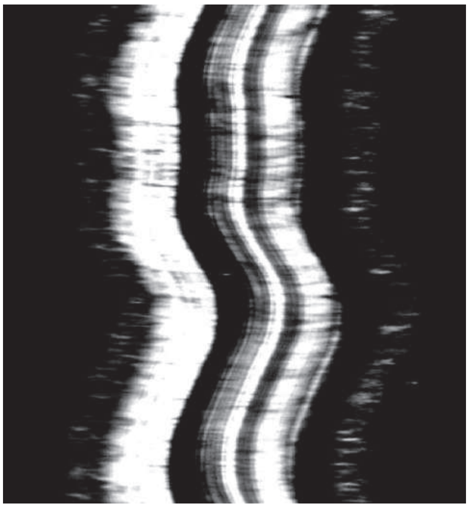


Figure 6-3: Side-lightening with aluminum disc

This particularity makes it very difficult to analyze with the Max derivative number 4 method. On the other hand, the optical flow reacts quite well, because it has a lot of different edges. I have got 2 different samples from the same disc:

- Aluminum0
- Aluminum1

As this disc was supposedly quite special, it was decided to check another aluminum sample. The Max derivative method is usually not used on those aluminum discs, but they are generally analyzed by the 3D probe system. A sample which had already been analyzed by the 3D probe was found, so that it is possible to compare the optical flow method to it. The sample is called:

- Aluminum4147

Memovox discs

The Memovox works with a stylus and groove, as most of the records. The particularity of this machine was that the stylus was moving at a constant linear velocity along the groove. This means that depending on where the stylus is compared to the radius, the disc does not rotate at the same speed [26, p. 2005].

The Memovox discs were made of soft plastic and were mostly radio or intelligence records from WWII. Their main particularity is their lack of any clear groove bottom. The resulting image is a set of lines, which makes the groove detection very complicated, as visible in Figure 6-4.

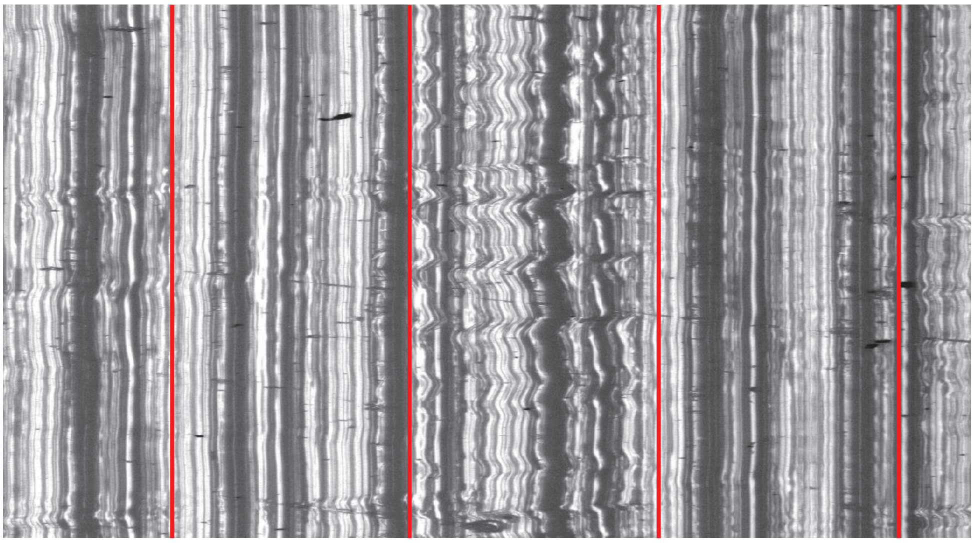


Figure 6-4: Memovox record with supposed tracks in red

Again, the optical flow method should work really well with that kind of samples. I have got 2 different samples from the same disc but from different side:

- Memovox0a
- Memovox0b

Phase samples

Some extra samples have been provided by another laboratory in Berkeley. Their acquisition system is based on measuring the phase difference to determine the height of a sample. In Figure 6-5, we can see a part of the resulting image.

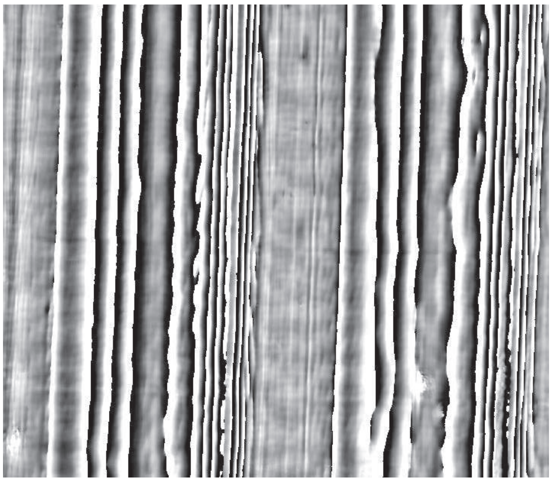


Figure 6-5: Image acquired by the phase system

The gray gradient represents the phase. A black pixel is 0° and a white corresponds to 359°. If the color jumps from black to white, then that means that the height does not change. As this system works with unwrapped phase, it is normal to get these results. Some tests were done on these kinds of samples with the optical flow algorithm but without success. The analyzed samples were not big. It is then possible that the sound present on the picture was just not long enough to be understandable.

Another point is that the image does not seem very logical at some parts. In Figure 6-6, we can see in the middle a clear jump from black to white. This one is fine. On the other hand, the left part seems to jump from black to white, but it does not do it on the whole sample. It is still not known if this is totally normal, or if it is due to the sample.

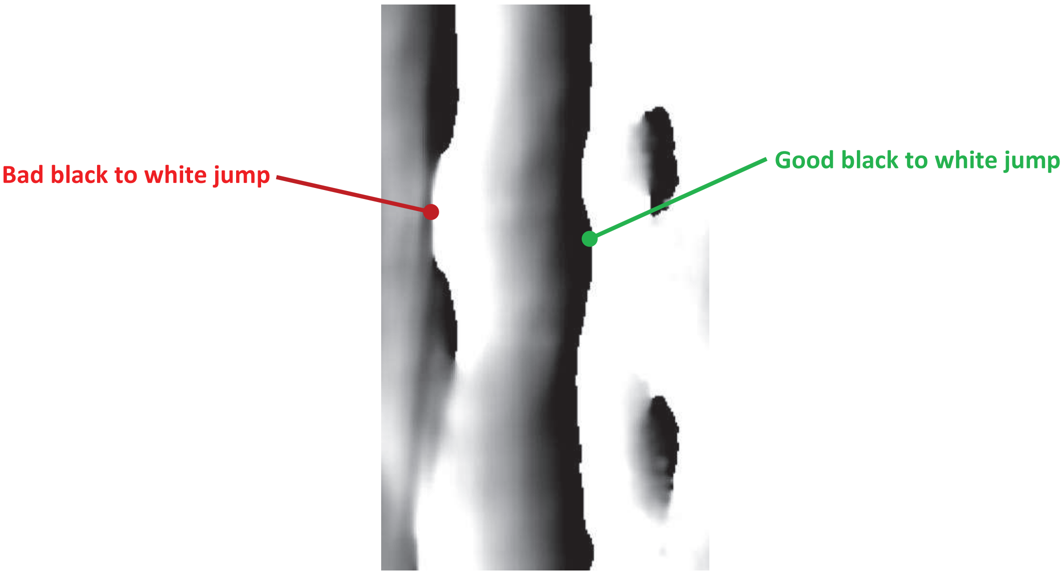


Figure 6-6: Zoom-in of the phase sample

Those samples will finally not be further analyzed.

6.3 Determination of the best optical flow method

6.3.1 Introduction

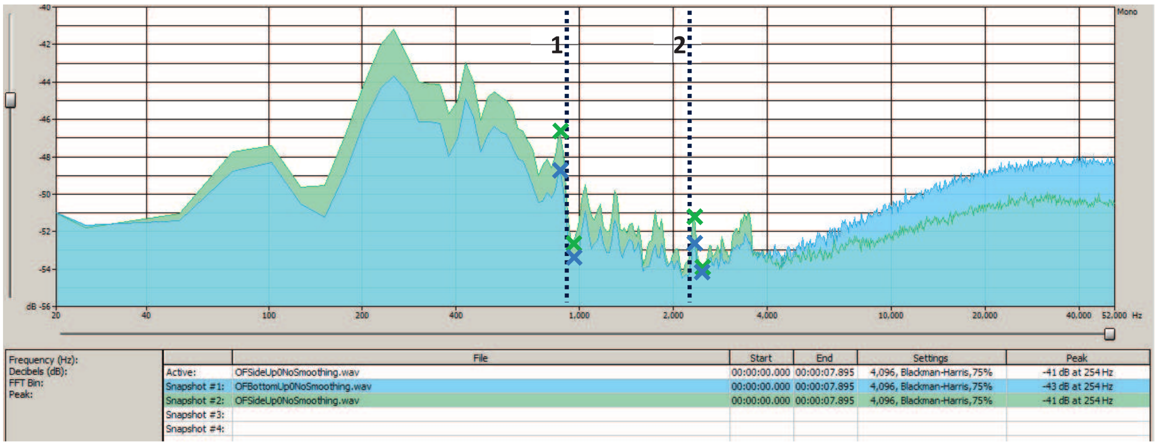
Those tests were mostly done on the music samples to try to determine which optical flow method with what parameters gives the best results. Different experiments were done with different types of imaging, up-sampling factor, or smoothing mode. At the end, 2 methods are kept for comparing with the Max derivative number 4 method in Section 6.5. The tests done in this section use the best parameters found in Chapter 6.4.6.

6.3.2 Type of images

With the optical flow, the quality of the sound changes a lot with the quality and type of the image. The more groove an image has, the better the optical flow works. In Figure 6-7, we can see a comparison between normal samples and side-lightening samples.

The tests were done with music samples, but silent samples were also used for comparison.

Music samples



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-7: Normal image in blue and side-lightening in green (LDTruck5)

Screenshots with other samples can be found in Appendix 10.1. The resulting values are written in the next tables.

LDTruck5	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	4.9	6	1.76	2.00	12%
Point 2	1.8	2.7	1.23	1.36	10%

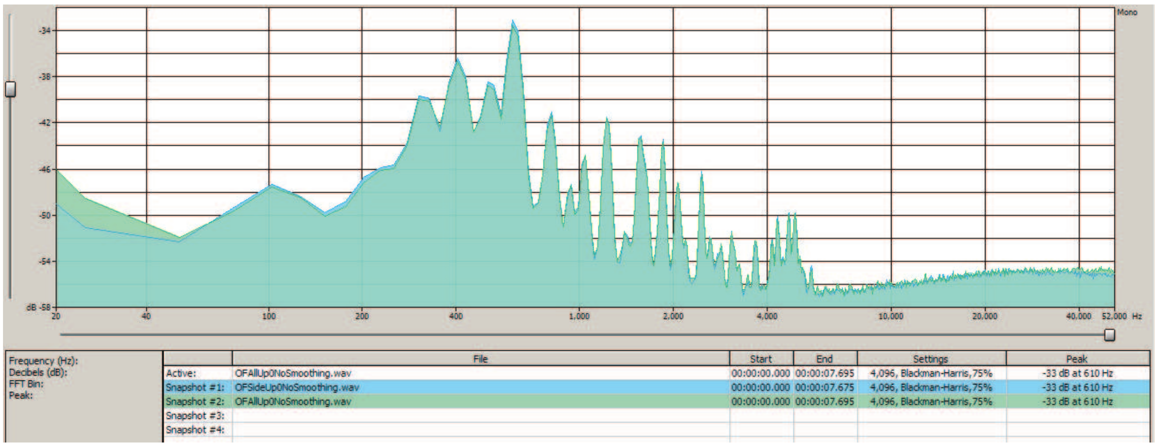
Nobody16	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	11.2	12.5	3.63	4.22	14%
Point 2	6.6	7.9	2.14	2.48	14%

SSB25	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	9.3	10	2.92	3.16	8%
Point 2	8.8	10.7	2.75	3.43	20%

DC1	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	9.8	12.1	3.09	4.03	23%
Point 2	7.3	9.7	2.32	3.05	24%

We clearly see that the side-lightening images increase the audio quality (between 10% and 20%). The improvement also depends a lot on the quality of the discs.

A comparison between side-lightening only (2 grooves) and side plus bottom lightening (3 grooves) can be seen in Figure 6-8. Both spectra are very similar. The valleys of the side-lightening only image seem however slightly steeper. The reason is probably due to the superposition effect (see Chapter 5.4.2 for more details about the superposition). Even if the picture was taken by trying to avoid it, there are still some parts of the groove that are superposed to each other.

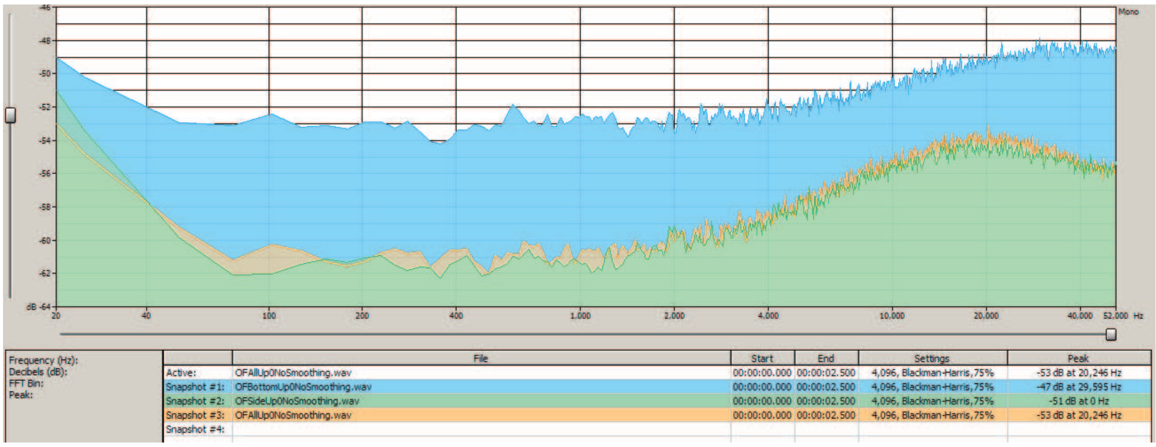


	File	Start	End	Settings	Peak
Active:	OFAllUp0NoSmoothing.wav	00:00:00.000	00:00:07.695	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #2:	OFAllUp0NoSmoothing.wav	00:00:00.000	00:00:07.695	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-8: Side-lightening only in blue and bottom plus side-lightening in green (DC1)

Silent samples

One test was also done on a silent sample (normal image, side-lightening image, and 3 groove image). The resulting spectrum can be seen in Figure 6-9. The noise level is lowered by about 8dB in the middle frequencies, which is a factor of 2.5. This really shows that side-lightening images work much better with the optical flow algorithm. Here again, we also see that the 3 groove image finally does not improve the result that much.



	File	Start	End	Settings	Peak
Active:	OFAllUp0NoSmoothing.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-53 dB at 20,246 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-47 dB at 29,595 Hz
Snapshot #2:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-51 dB at 0 Hz
Snapshot #3:	OFAllUp0NoSmoothing.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-53 dB at 20,246 Hz
Snapshot #4:					

Figure 6-9: Silent samples with normal image in blue, side-lightening in green, bottom plus side-lightening in orange (DC0)

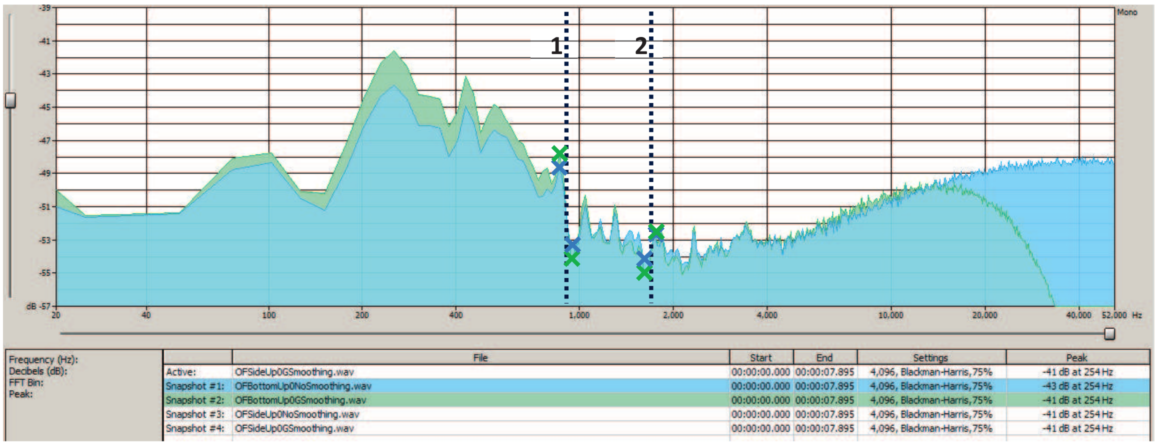
6.3.3 Smoothing

The smoothing question can be discussed for a long time. The most important fact to point out is that archives prefer to have a lower quality without any smoothing. Because smoothing the image creates a low-pass filter on the audio, this filter could potentially remove some important information.

On the other hand, the optical flow works better when the image was smoothened before. This means than smoothing the image before makes the optical flow algorithm react much better in the middle frequencies (1 to 6kHz).

No smoothing vs Gauss filter

The biggest differences can be seen when no smoothing is compared to the Gauss filter. We see in Figure 6-10 that the result is globally better with smoothing, but the higher frequencies components over 15kHz are attenuated a lot as expected.



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-43 dB at 254 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #4:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz

Figure 6-10: Effect of the smoothing on normal images without smoothing in blue and with Gaussian filter in green (LDTruck5)

Screenshots with other samples can be found in Appendix 10.2. The resulting values are written in the next tables for normal images. The difference can be quite important depending on the samples.

LDTruck5	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	4.8	6.4	1.74	2.09	17%
Point 2	1.7	2.3	1.22	1.30	7%

Nobody16	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	7.3	8.7	2.32	2.72	15%
Point 2	6.5	7.2	2.11	2.29	8%

SSB25	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	12.3	12.8	4.12	4.37	6%
Point 2	8.9	9.5	2.79	2.99	7%

DC1	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	6.9	8.2	2.21	2.57	14%
Point 2	7.2	9	2.29	2.82	19%

When we use side-lightening images, smoothing them seem to have less effect than on ordinary images. There is still some improvement depending on the samples, but the quality enhancement is smaller. Using side-lightening without smoothing can give good results, avoiding the low-pass filter effect on the audio signal.

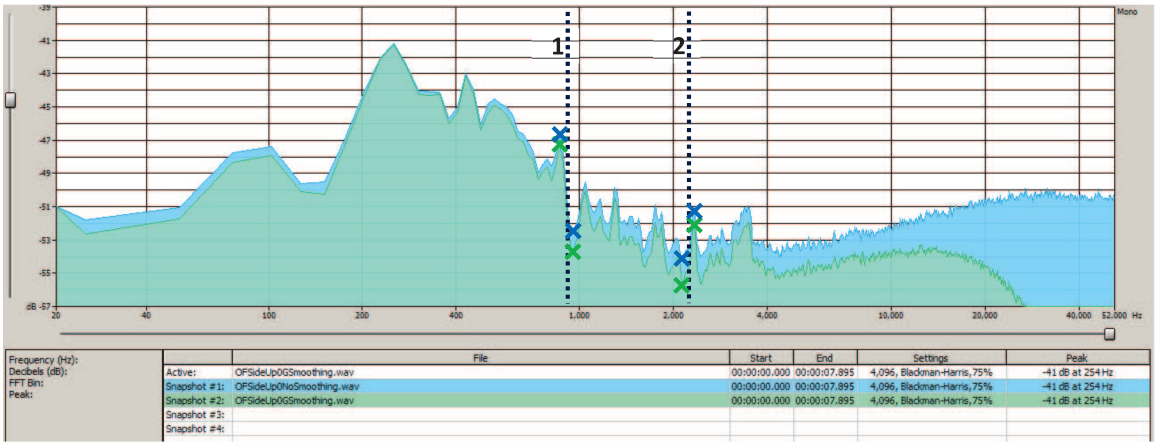
The spectra of the other samples can be found in Appendix 0. The results are given in the tables below.

LDTruck5	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	6.1	6.4	2.02	2.09	3%
Point 2	3	3.5	1.41	1.50	6%

Nobody16	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	8.3	9.3	2.60	2.92	11%
Point 2	7.7	8	2.43	2.51	3%

SSB25	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	7.8	7.9	2.45	2.48	1%
Point 2	10.8	11.2	3.47	3.63	5%

DC1	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	8.1	9.7	2.54	3.05	17%
Point 2	9.7	10.6	3.05	3.39	10%



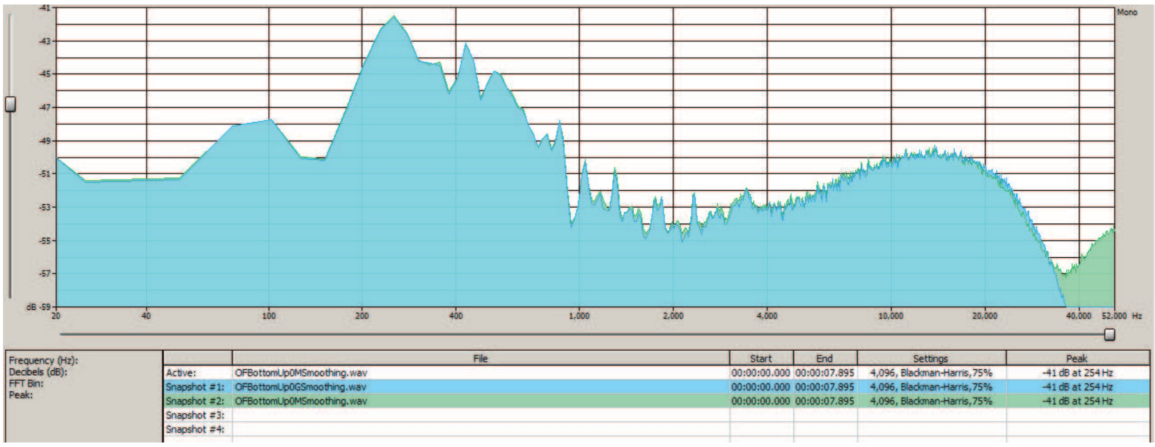
Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #2:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-11: Effect of the smoothing on side-lightening images without smoothing in blue and with Gaussian filter in green (LDTruck5)

Gauss filter vs median filter

The Gauss filter and median filter have more or less the same effect on the sound. We can see in Figure 6-12 that the higher frequencies are less attenuated with the second one. The effect of both filters may also change depending on the quality of the images.

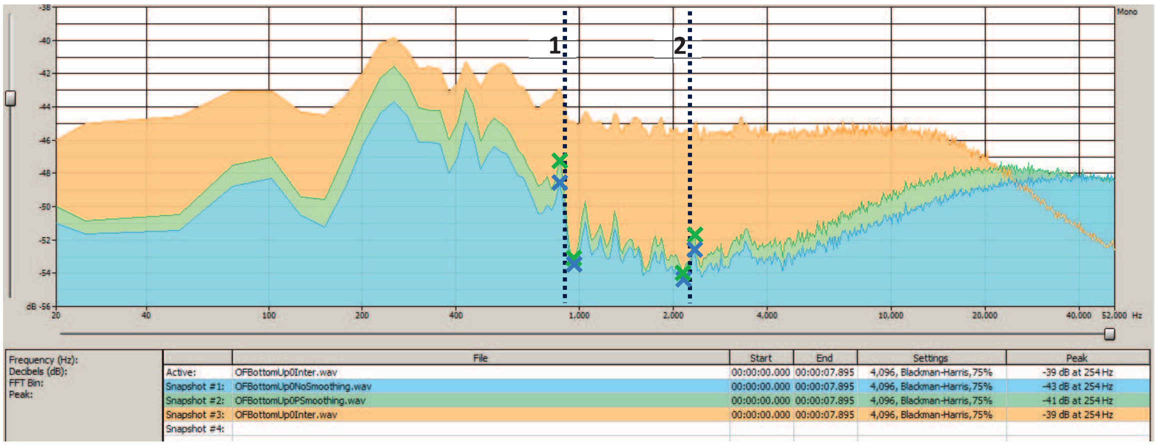


	File	Start	End	Settings	Peak
Active:	OFBottomUp0MSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #1:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #2:	OFBottomUp0MSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-12: Gauss filter in blue and median filter in green (LDTruck5)

Partial smoothing and interpolation

For normal images with the groove bottom only as in Figure 6-13, we clearly see that the interpolation method does not work. Because the algorithm is not perfect at all, too much interpolation is done and it affects the signal quality. On the other hand, the partial smoothing increases a bit the quality. The enhancement is very small, but it does not create as much low-pass effect in higher frequencies as a complete smoothing.



	File	Start	End	Settings	Peak
Active:	OFBottomUp0Inter.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-39 dB at 254 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	OFBottomUp0PSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #3:	OFBottomUp0Inter.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-39 dB at 254 Hz
Snapshot #4:					

Figure 6-13: Normal images without smoothing in blue, with partial smoothing in green, and with interpolation in orange (LDTruck5)

Other spectra can be found in Appendix 10.2. The results are given in the following tables.

LDTruck5	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	4.8	5.9	1.74	1.97	12%
Point 2	2	2.4	1.26	1.32	5%

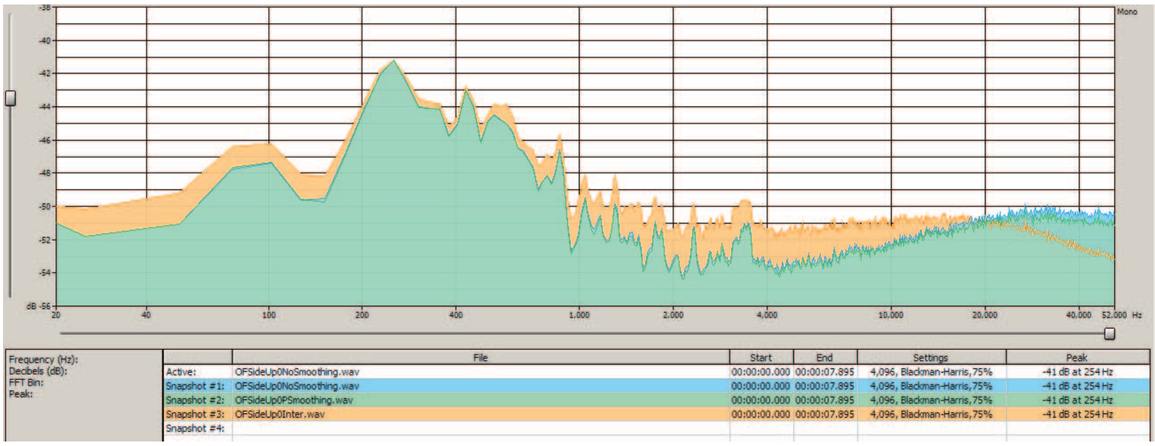
Nobody16	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	11.2	12.6	3.63	4.27	15%
Point 2	6.4	6.9	2.09	2.21	6%

SSB25	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	9.4	9.6	2.95	3.02	2%
Point 2	9	9.2	2.82	2.88	2%

DC1	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	14.1	15	5.07	5.62	10%
Point 2	8.1	8.5	2.54	2.66	5%

The partial smoothing ameliorates the quality, but it also depends on the disc. For example the SSB25 sample has almost no amelioration. The partial smoothing also tends to enhance the quality more in the lower frequencies around 1kHz.

With side-lightening images, there are usually no cracks on the images. Or it is very unlikely to get a scratch that damages all the grooves. This is why the results are very similar without smoothing or with partial smoothing. The interpolation method is still not right either. Other samples can be seen in Appendix 10.5 and they also show that the partial smoothing does not bring a better quality for side-lightening images.



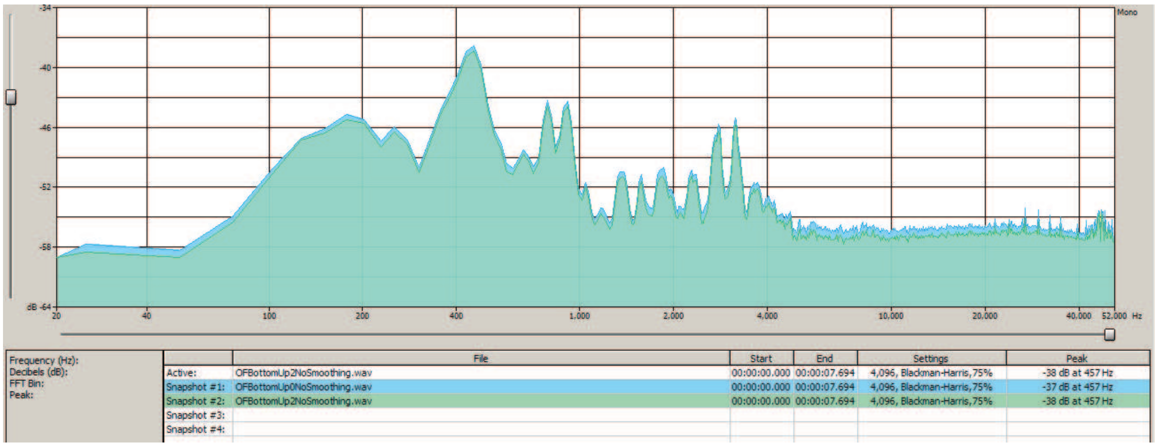
	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #2:	OFSideUp0PSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #3:	OFSideUp0Inter.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris, 75%	-41 dB at 254 Hz
Snapshot #4:					

Figure 6-14: Side-lightening images without smoothing in blue, with partial smoothing in green, and with interpolation in orange (LDTruck5)

6.3.4 Up-sampling

The up-sampling order was a useful parameter at the beginning of the research about optical flow. It gave a better quality to the single groove images which did not have enough gray pixels. Now it has almost no effect anymore. The different image processing used along with the optical flow makes the algorithm good enough without using any up-sampling.

In Figure 6-15, we can see a comparison between normal images (single groove) with and without up-sampling. The only remaining difference is a matter of volume adjustment.



	File	Start	End	Settings	Peak
Active:	OFBottomUp2NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-37 dB at 457 Hz
Snapshot #2:	OFBottomUp2NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-15: Comparison without up-sampling in blue and with up-sampling in green (SSB25)

6.3.5 Discussion

After looking at the test results, one important point shows up. The different parameters and optical flow methods vary a lot the results, but the quality of the samples themselves changes a lot the reaction of the algorithms. The image acquisition finally plays an important role to get a better audio quality.

The side-lightening images globally give better results and do not need as much smoothing as normal images to get a good audio quality. Both median and Gaussian filters give identical results. On the other hand, the interpolation algorithm does not work correctly because the results are habitually worsened. The up-sampling was a useful tool at the beginning of the project, but it does not help that much anymore.

6.3.6 Best methods

Based on the previous tests, the optical flow with side-lightening images gives the best results. This method also does not need smoothing, which is a good point to avoid any low-pass filter effect.

For normal images, the best results can be found when using the Gaussian filter.

Those 2 methods will be used to be compared to the best traditional method. We should always keep in mind that if we want to avoid the low-pass filter effect, side-lightening images are way better to process.

6.4 Determination of the other parameters

6.4.1 Introduction

The determination of the other parameters was done only with one sample. It was completed to determine which parameters give the better results with the optical flow method. No values will be taken from the spectrum, but a general appreciation will be done to define the best parameters.

The sample used for these test is the **side-lightening** version of the "LDTruck5" sample. The side-lightening images usually result in more velocity vectors and modifying the parameters would change the resulting signal more than normal images. The resulting audio samples for these tests are stored in the folder labeled "LDTruck5/Audio/Parameters". The following parameters are chosen by default:

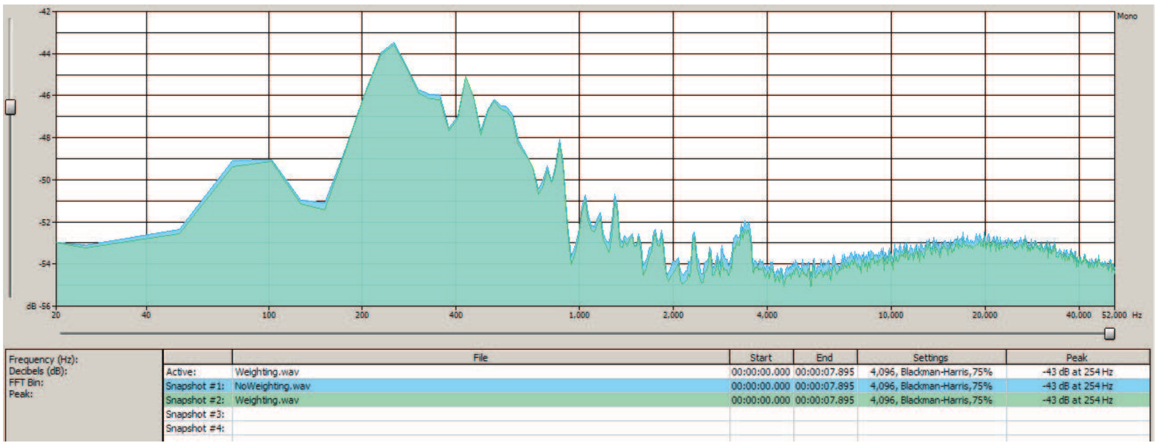
- No up-sampling
- Lucas-Kanade algorithm
- No smoothing
- Normal averaging
- Sobel weighting is deactivated

- Sobel level at 60%

This sample has a big peak around 1kHz and it helps for the comparison.

6.4.2 Sobel weighting

The resulting audio files for this test are “NoWeighting” and “Weighting”. The results are very similar, but it seems that the weighting slightly improves some of the middle frequencies. The signal-to-noise ratio is about 0.2dB better for the weighting version.



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

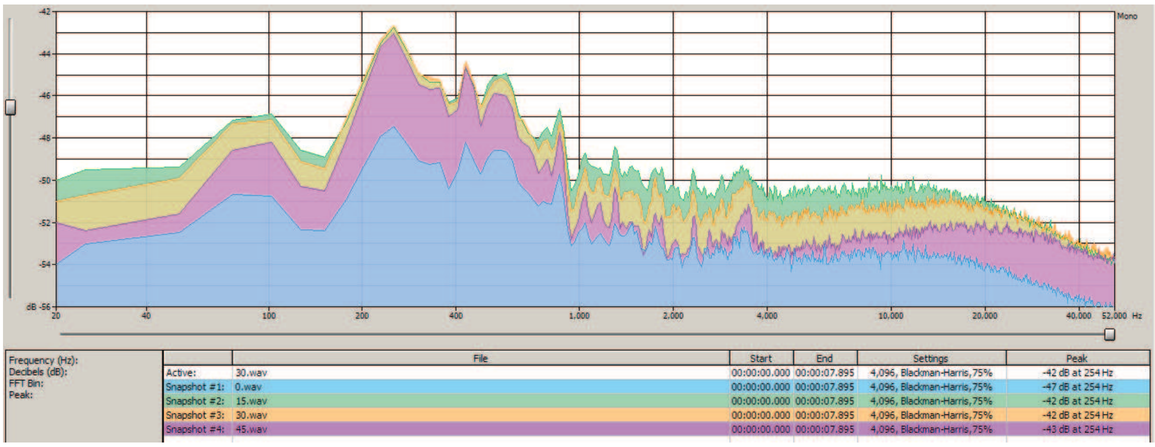
	File	Start	End	Settings	Peak
Active:	Weighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #1:	NoWeighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	Weighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-16: Comparison without Sobel weighting in blue and with Sobel weighting in green (LDTruck5)

6.4.3 Sobel level

The number contained in the audio filenames corresponds to the level. The tests have been with the following levels: 0, 15, 30, 45, 60, 75, and 90. Higher values do not make sense, because only a few velocity vectors would be taken into account.

In the figure below, we can see the 4 first levels. The main difference is difference of amplitude. If we compare the peak at around 1kHz, we can see that the absolute difference between the peak and the valley close to it is getting bigger with the Sobel level.

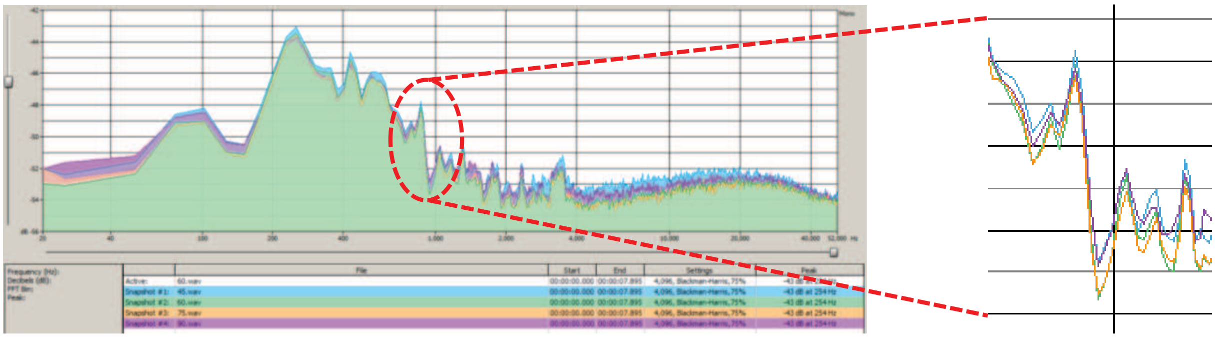


Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	30.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-42 dB at 254 Hz
Snapshot #1:	0.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-47 dB at 254 Hz
Snapshot #2:	15.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-42 dB at 254 Hz
Snapshot #3:	30.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-42 dB at 254 Hz
Snapshot #4:	45.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz

Figure 6-17: Comparison with Sobel level at 0 (blue), 15 (green), 30 (orange), and 45 (violet) (LDTruck5)

In Figure 6-18, we can see the 4 last levels. There is almost no difference, which means that high Sobel levels tend to give the same results. If we zoom around 1kHz as on the picture to the right, we see that the yellow and green traces have the biggest difference between the peak and the valley.



	File	Start	End	Settings	Peak
Active:	60.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #1:	45.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	60.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #3:	75.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #4:	90.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz

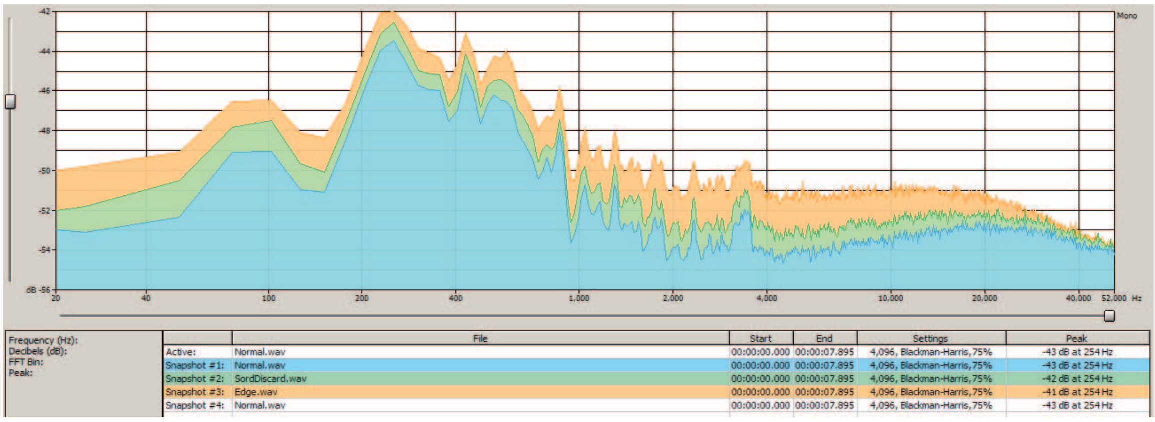
Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

Figure 6-18: Comparison with Sobel level at 45 (blue), 60 (green), 75 (orange), and 90 (violet) (LDTruck5)

A default value for the Sobel level would be between **60** and **75**. Another point to take into account is that the optimal level can change from one picture to another depending on the contrast. The grayscale gradient is also not the same for every disc. This is why doing tests with different values will help to determine the best one.

6.4.4 Averaging method

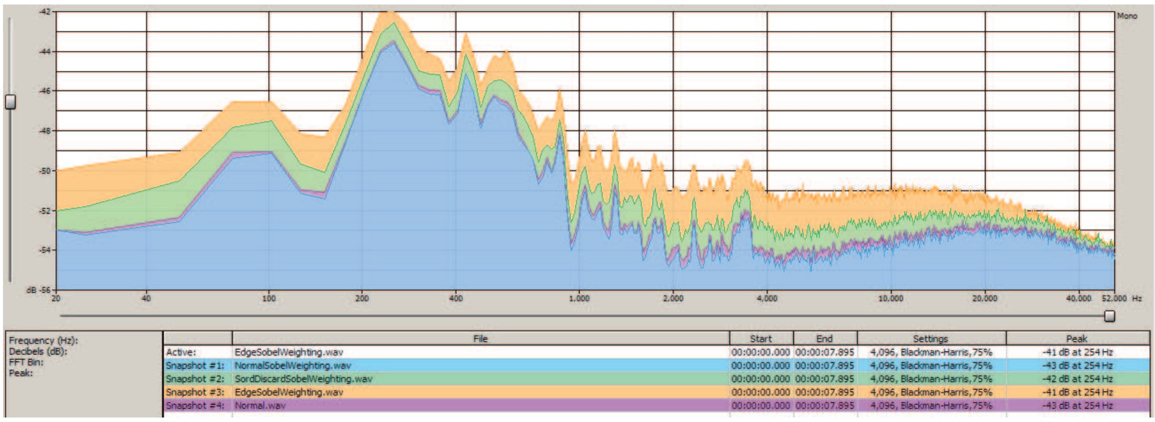
The resulting audio files for this test are “Normal”, “SordDiscard”, “Edge”, corresponding to the 3 different methods of averaging. We see in the figure below that the normal averaging method is the one that gives the best result. It was then decided to do the same tests, but the Sobel weighting parameter would be enabled. The result can be seen in Figure 6-20.



	File	Start	End	Settings	Peak
Active:	Normal.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #1:	Normal.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	SordDiscard.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-42 dB at 254 Hz
Snapshot #3:	Edge.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #4:	Normal.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz

Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

Figure 6-19: Normal averaging in blue, sort and discard in green, and edge in orange (without Sobel weighting) (LDTruck5)



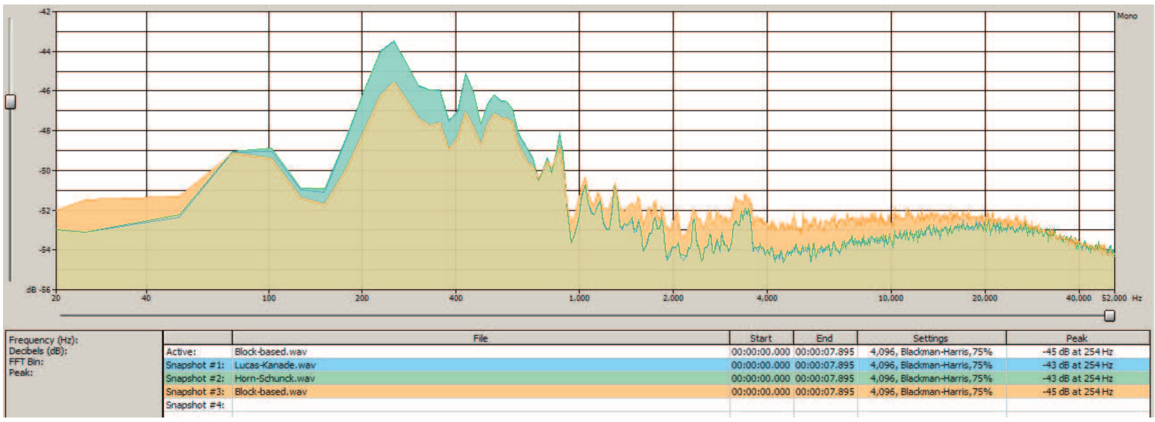
	File	Start	End	Settings	Peak
Active:	EdgeSobelWeighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #1:	NormalSobelWeighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	SordDiscardSobelWeighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-42 dB at 254 Hz
Snapshot #3:	EdgeSobelWeighting.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #4:	Normal.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz

Figure 6-20: Normal averaging in blue, sort and discard in green, edge in orange (with Sobel weighting), and normal averaging without Sobel weighting in violet (LDTruck5)

The normal averaging method still seems to give the best results. On the last figure, the blue spectrum is a bit better than the pink one, suggesting that using the Sobel weighting increases the audio quality.

6.4.5 Optical flow algorithm

The resulting audio files for this test are "Lucas-Kanade", "Horn-Schunck", and "Block-based". In Figure 6-21, we clearly see that the block-based method is the worst one (yellow spectrum). Both Lucas-Kanade and Horn-Schunck algorithms give on the other hand exactly the same spectrum.



	File	Start	End	Settings	Peak
Active:	Block-based.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-45 dB at 254 Hz
Snapshot #1:	Lucas-Kanade.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #2:	Horn-Schunck.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-43 dB at 254 Hz
Snapshot #3:	Block-based.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-45 dB at 254 Hz
Snapshot #4:					

Figure 6-21: Comparison between Lucas-Kanade optical flow method in blue, Horn-Schunck in green, and block-based in orange (LDTruck5)

6.4.6 Best parameters

After doing these few tests, the following parameters will be chosen to execute the main tests:

- Sobel weighting is activated
- The Sobel level is set to 60%
- Normal averaging method
- The Lucas-Kanade optical flow algorithm is used

It is important to point out that those ideal parameters can change from one audio sample to another and that it is always useful to make some extra tests to find out the ones which result in the best sound quality.

6.5 Comparing the optical flow methods to the Max derivative number 4

6.5.1 Introduction

After all the tests done in the previous chapters, the best optical flow methods were found and can finally be compared to the best traditional method, which is the Max derivative number 4. For most of the tests in this chapter, the 3 following cases will be used for comparison:

- Normal images with the Max derivative number 4 algorithm (**blue spectrum**)
- Normal images with the optical flow algorithm with Gaussian smoothing (**green spectrum**)
- Side-lightening images with the optical flow algorithm without smoothing (**yellow spectrum**)

If nothing is specified, the spectra colors correspond to the ones written above. These tests are done on different kind of samples. The Chapter 6.5.4 treats special samples. Those samples usually do not give good results with the Max derivative number 4 and are therefore more difficult to compare. Extra explanation will be given in that chapter.

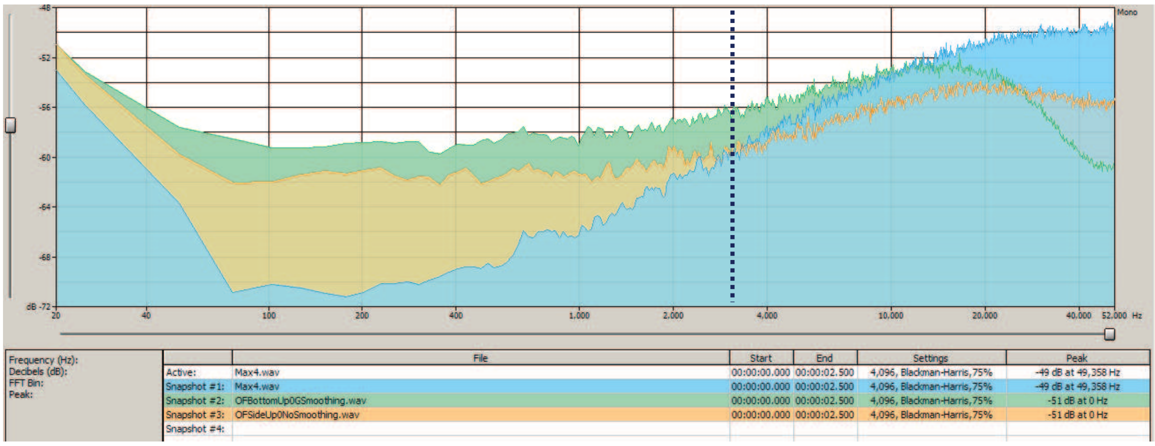
6.5.2 Silent samples

The silent samples give a good overview on how the algorithm reacts with noise level over the frequencies. Different measurements will be done on the different samples in order to try to quantify the quality of the algorithms. The different measurements will be the following:

- Level difference compared to the Max derivative number 4 at 400Hz.
- Level difference compared to the Max derivative number 4 at 20kHz.
- Frequency at each both Max derivative number 4 and optical flow with side-lightening images methods have the same noise level.

For the optical flow with Gaussian smoothing, it is harder to compare due to the low-pass filter effect. This is why the last measurement will not be done with the green spectrum. A summary of the measurements can be found at the end of this chapter.

DC0 silent sample



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

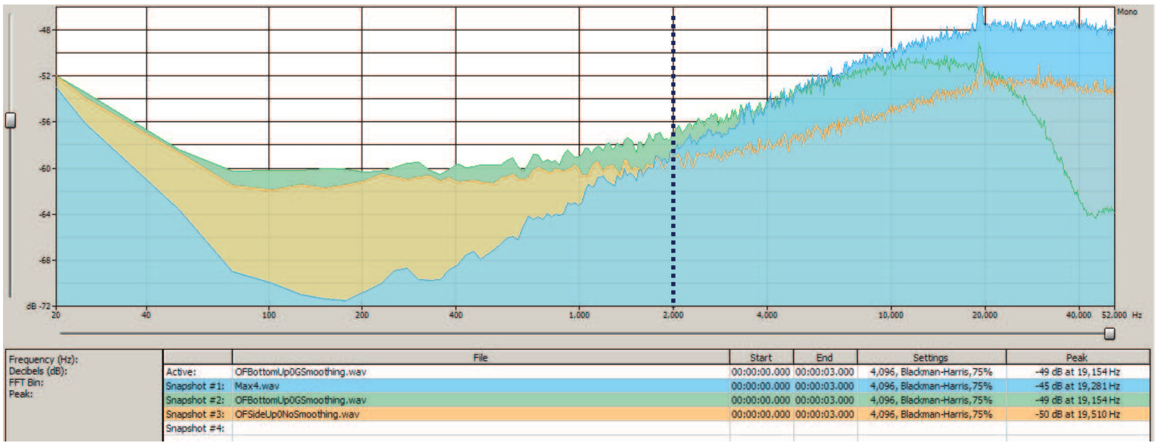
	File	Start	End	Settings	Peak
Active:	Max4.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-49 dB at 49,358 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-49 dB at 49,358 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-51 dB at 0 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:02.500	4,096, Blackman-Harris,75%	-51 dB at 0 Hz
Snapshot #4:					

Figure 6-22: DC0 silent sample

The Max derivative number 4 has the same level as the side-lightening images at about 3.5kHz. We can see the different levels in the table below.

Frequency [Hz]	Max derivative 4 [dB]	Normal images [dB]	Side-lightening images [dB]
400	-69	-58.7	-61.5
20000	-50.7	-53.2	-54.3

Kansas1 silent sample



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

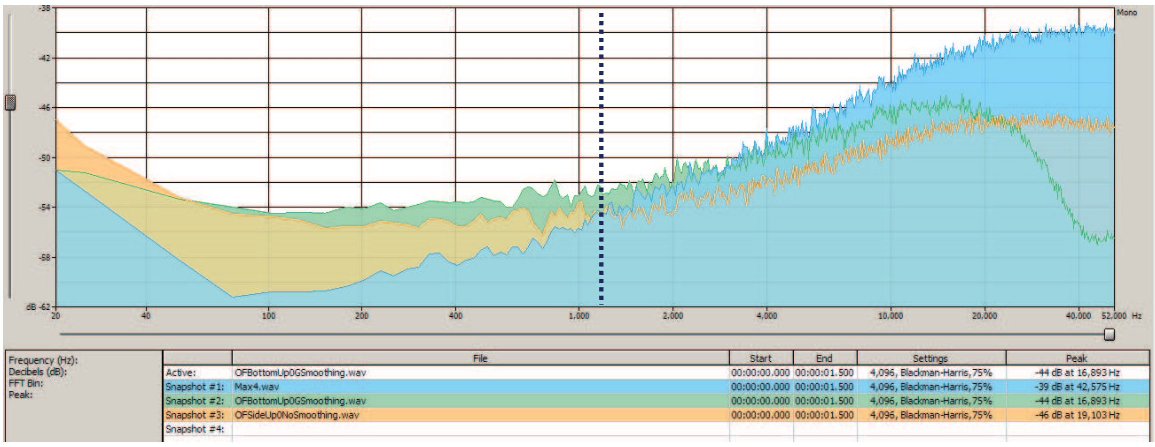
	File	Start	End	Settings	Peak
Active:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:03.000	4,096, Blackman-Harris,75%	-49 dB at 19,154 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:03.000	4,096, Blackman-Harris,75%	-45 dB at 19,281 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:03.000	4,096, Blackman-Harris,75%	-49 dB at 19,154 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:03.000	4,096, Blackman-Harris,75%	-50 dB at 19,510 Hz
Snapshot #4:					

Figure 6-23: Kansas1 silent sample

The Max derivative number 4 has the same level as the side-lightening images at about 2kHz. We can see the different levels in the table below.

Frequency [Hz]	Max derivative 4 [dB]	Normal images [dB]	Side-lightening images [dB]
400	-68.4	-59.7	-61.2
20000	-47.7	-51.3	-52.6

LDTruck0 silent sample



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:01.500	4,096, Blackman-Harris,75%	-44 dB at 16,893 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:01.500	4,096, Blackman-Harris,75%	-39 dB at 42,575 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:01.500	4,096, Blackman-Harris,75%	-44 dB at 16,893 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:01.500	4,096, Blackman-Harris,75%	-46 dB at 19,103 Hz
Snapshot #4:					

Figure 6-24: LDTruck0 silent sample

The Max derivative number 4 has the same level as the side-lightening images at about 1.2kHz. We can see the different levels in the table below.

Frequency [Hz]	Max derivative 4 [dB]	Normal images [dB]	Side-lightening images [dB]
400	-58.5	-53.6	-55.5
20000	-40.8	-46.1	-46.9

Comparison

We can see that the Max derivative algorithm gives better results in the lower frequencies, but it becomes worse in higher frequencies. On the few samples, the average frequency when the optical flow algorithm with side-lightening images becomes better is around 2.2kHz. This value can vary quite a lot depending on the samples.

In the table below, we can see the differences between the Max derivative 4 and the 2 optical flow methods. A negative value means that the Max derivative has a lower noise level (and vice-versa).

Samples	Difference between Max derivative 4 and normal images [dB]		Difference between Max derivative 4 and side-lightening images [dB]	
	400Hz	20000kHz	400Hz	20000kHz
DC0	-10.3	2.5	-7.5	3.6
Kansas1	-8.7	3.6	-7.2	4.9
LDTruck0	-4.9	5.3	-3	6.1

We see that the LDTruck0 sample seems better compared to the other samples (DC0 gives the worse result with the optical flow). Let us compare the images to see where this difference comes from.

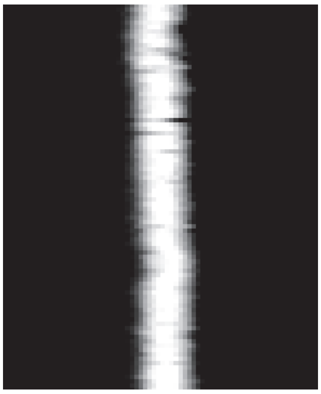


Figure 6-25: DC0 normal image

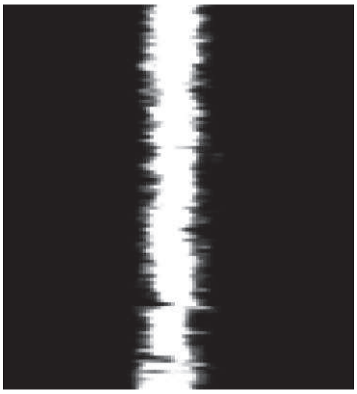


Figure 6-26: LDTruck0 normal image

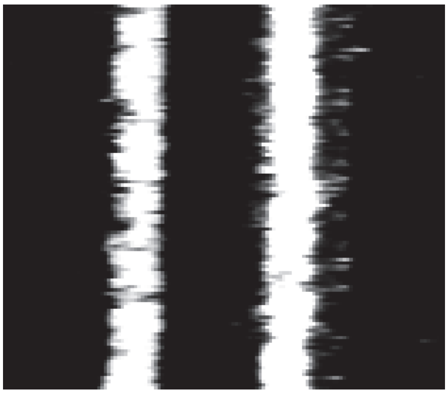


Figure 6-27: LDTruck0 side-lightening image

The DC0 sample in Figure 6-25 seems smoother than the LDTruck0 sample in Figure 6-26, which is very noisy. This may come from the acquisition process. Depending on the quality of the groove, one algorithm seems to work better than the other one. If the acquired image is noisier, then the optical flow method will tend to react better to the noise than the Max derivative algorithm.

When listening to the audio file, the noise level seems very similar. As expected, the noise from the optical flow method sounds lower in the frequencies than with the Max derivative.

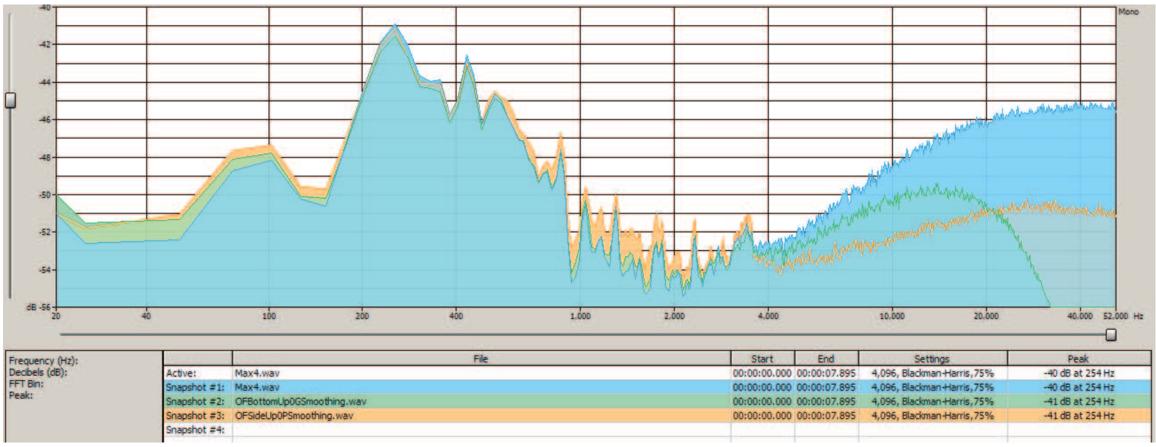
6.5.3 Music samples

When comparing the best methods together, it really seems that they mostly give the same results. The spectra for different samples can be found in the Figure 6-28 below and in Appendix 10.6. In the middle frequencies (lower than 3kHz), all the methods almost have the same frequency response and the difference are too small to be quantifiable. All the samples work pretty well with both algorithms.

It is getting interesting in the higher frequencies (higher than 3kHz). As the noise level is usually lower with the optical flow methods (see previous chapter for more details), the higher frequencies become better than with the Max derivative number 4 algorithm. The response of the Max derivative algorithm also varies a lot in the higher frequencies depending on the kind of samples.

When listening to the audio files, they really sound similar. It is very difficult to hear the difference and the people ears do not have the same frequency response. The noise seems a bit louder with the Max derivative algorithm.

However, the sound of the noise is different from one algorithm to the other. As it has already been heard with silent samples, the noise from the optical flow method seems lower in the frequencies.

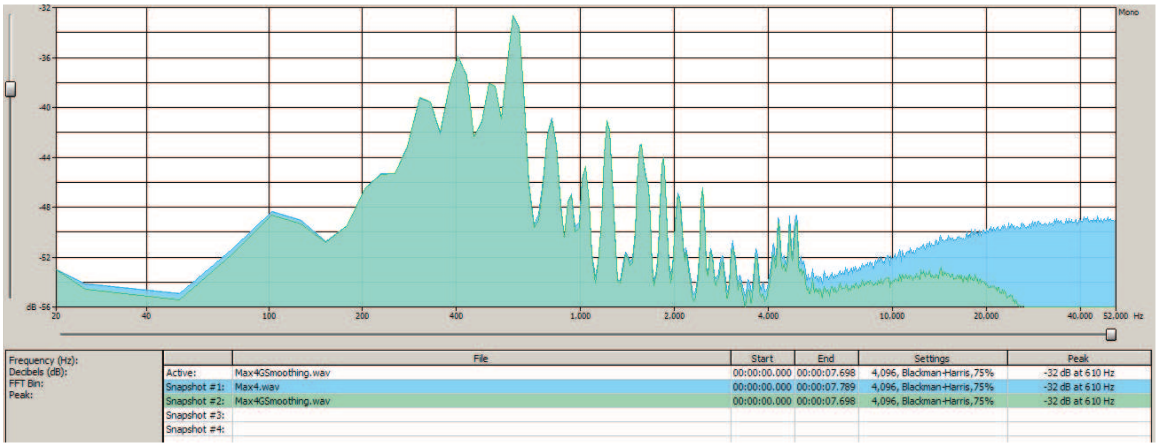


	File	Start	End	Settings	Peak
Active:	Max4.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-40 dB at 254 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-40 dB at 254 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #3:	OFSideUp0PSmoothing.wav	00:00:00.000	00:00:07.895	4,096, Blackman-Harris,75%	-41 dB at 254 Hz
Snapshot #4:					

Figure 6-28: LDTruck5 music sample

In Figure 6-29, we can see a comparison between applying and not applying the Gaussian filter to the image with the Max derivative number 4 algorithm. As expected, the higher frequencies are cut off. At around 4kHz, the green spectrum looks better and less noisy. This shows that smoothing the image before processing can increase the sound quality at the expanse of losing the higher frequencies.

The same result without the low-pass filter effect can be reached by using side-lightening images. The higher frequencies are smoothened because the image contains more information, and the noise which is random is averaged.



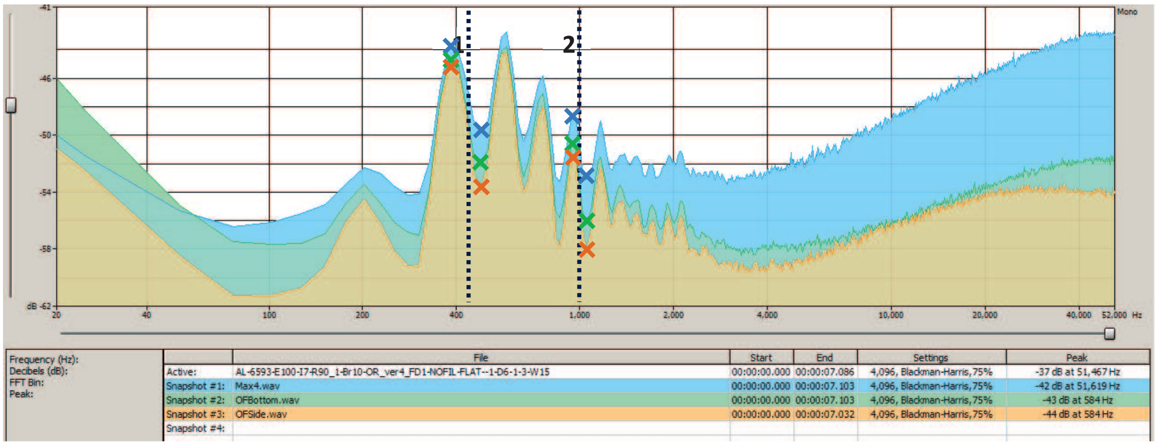
	File	Start	End	Settings	Peak
Active:	Max4GSmoothing.wav	00:00:00.000	00:00:07.698	4,096, Blackman-Harris,75%	-32 dB at 610 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris,75%	-32 dB at 610 Hz
Snapshot #2:	Max4GSmoothing.wav	00:00:00.000	00:00:07.698	4,096, Blackman-Harris,75%	-32 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-29: Comparison of the Max derivative method without smoothing in blue and with Gaussian filter in green (DC1)

6.5.4 Special samples

Aluminum discs

In Figure 6-30, we can see the spectrum of the first aluminum disc sample. We can clearly see that the optical flow methods give better results than the Max derivative algorithm. The noise in the higher frequencies is much lower.



	File	Start	End	Settings	Peak
Active:	AL-6593-E100-I7-R90_1-Br10-OR_ver4_FD1-NOFIL-FLAT--1-D6-1-3-W15	00:00:00.000	00:00:07.086	4,096, Blackman-Harris,75%	-37 dB at 51,467 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.103	4,096, Blackman-Harris,75%	-42 dB at 51,619 Hz
Snapshot #2:	OFBottom.wav	00:00:00.000	00:00:07.103	4,096, Blackman-Harris,75%	-43 dB at 584 Hz
Snapshot #3:	OFSide.wav	00:00:00.000	00:00:07.032	4,096, Blackman-Harris,75%	-44 dB at 584 Hz
Snapshot #4:					

Figure 6-30: Aluminum0 sample

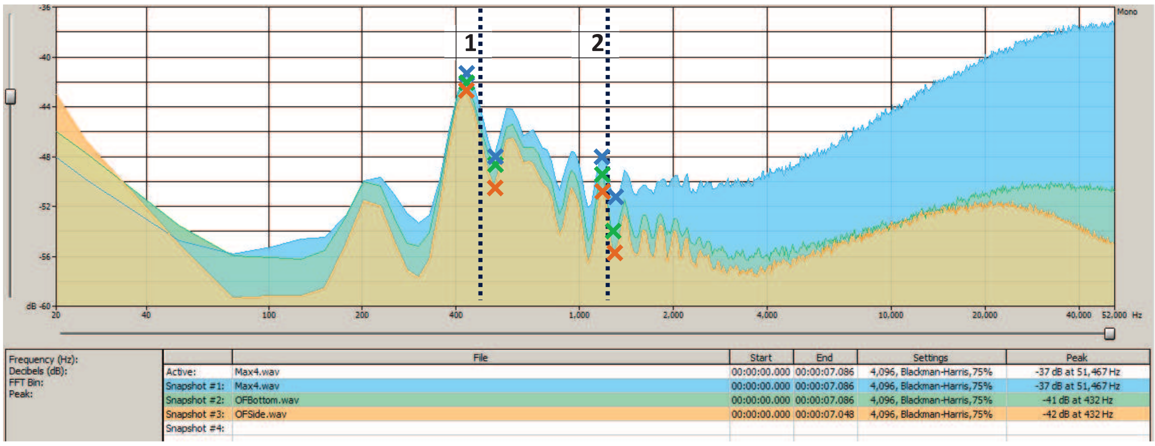
The table below shows the different measurements made on the spectrum in Figure 6-30.

Aluminum0	SNR Blue [dB]	SNR Green [dB]	SNR Orange [dB]	SNR Blue (linear)	SNR Green (linear)	SNR Orange (linear)
Point 1	6	7.6	8.6	2.00	2.40	2.69
Point 2	4.6	5.7	6.5	1.70	1.93	2.11

The relative difference has been calculated between the Max derivative algorithm and optical flow with normal lightening to the optical flow with side-lightening optical flow method. The results are the following:

- Relative difference between Max derivative and side-lightening optical flow:
  - Point 1: **26%**
  - Point 2: **20%**
- Relative difference between normal lightening and side-lightening optical flow:
  - Point 1: **11%**
  - Point 2: **9%**

As expected, the side-lightening gives better result. The quality enhancement is quite important. The Aluminum1 sample gives similar results. The spectrum looks similar with the roll-off in the higher frequencies for the side-lightening images.



	File	Start	End	Settings	Peak
Active:	Max4.wav	00:00:00.000	00:00:07.086	4,096, Blackman-Harris,75%	-37 dB at 51,467 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.086	4,096, Blackman-Harris,75%	-37 dB at 51,467 Hz
Snapshot #2:	OFBottom.wav	00:00:00.000	00:00:07.086	4,096, Blackman-Harris,75%	-41 dB at 432 Hz
Snapshot #3:	OFSide.wav	00:00:00.000	00:00:07.048	4,096, Blackman-Harris,75%	-42 dB at 432 Hz
Snapshot #4:					

Figure 6-31: Aluminum1 sample

The table below shows the different measurements made on the spectrum in Figure 6-31.

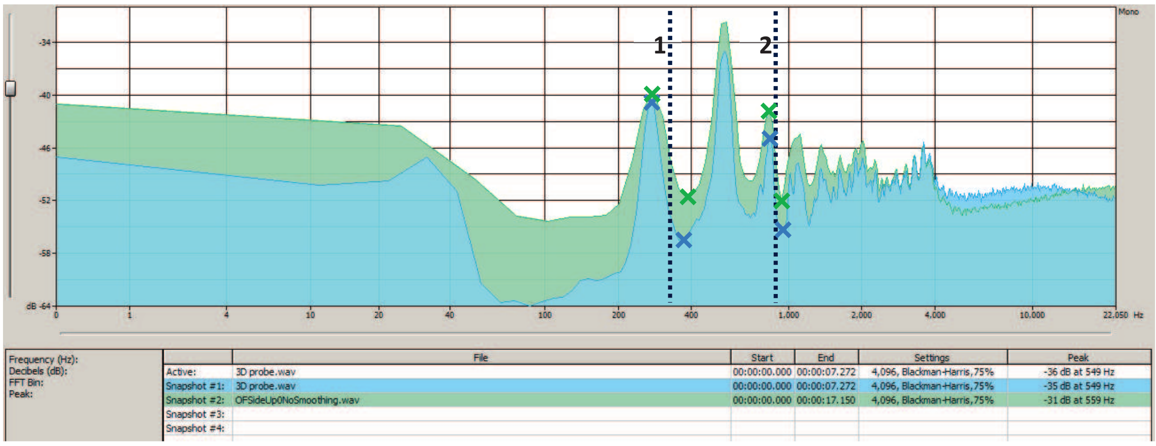
Aluminum1	SNR Blue [dB]	SNR Green [dB]	SNR Orange [dB]	SNR Blue (linear)	SNR Green (linear)	SNR Orange (linear)
Point 1	6.7	6.8	7.9	2.16	2.19	2.48
Point 2	3.5	4.7	5.7	1.50	1.72	1.93

The same relative differences as for the Aluminum0 sample were calculated:

- Relative difference between Max derivative and side-lightening optical flow:
  - Point 1: **13%**
  - Point 2: **22%**
- Relative difference between normal lightening and side-lightening optical flow:
  - Point 1: **12%**
  - Point 2: **11%**

Those results are quite good, but the disc was a special sample with an unusual groove shape. The classical aluminum discs are in general analyzed with the 3D probe. The Aluminum4147 sample was found. This sample had been acquired by the 3D system and processed by the PRISM software. The file labeled "3D probe.wav" is the resulting audio signal from it. This sample was also analyzed by the optical flow method on side-lightening images of the discs.

As expected, the PRISM software gives better results when handling traditional aluminum discs. In Figure 6-32, we can see the spectra of both 3D probe and optical flow algorithms.



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	3D probe.wav	00:00:00.000	00:00:07.272	4,096, Blackman-Harris, 75%	-36 dB at 549 Hz
Snapshot #1:	3D probe.wav	00:00:00.000	00:00:07.272	4,096, Blackman-Harris, 75%	-35 dB at 549 Hz
Snapshot #2:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:17.150	4,096, Blackman-Harris, 75%	-31 dB at 559 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-32: Aluminum4147 sample with 3D probe in blue and optical flow in green

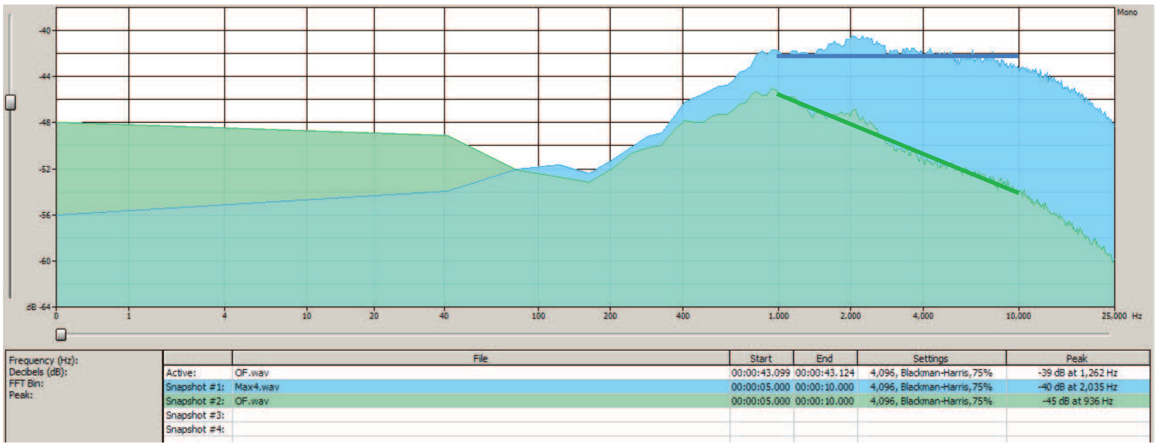
Al4147	SNR Blue [dB]	SNR Green [dB]	SNR Blue (linear)	SNR Green (linear)	Relative Difference (%)
Point 1	15.9	11.8	6.24	3.89	60%
Point 2	10.4	10.1	3.31	3.20	4%

The difference is huge in the lower frequencies. Around 1kHz and higher, there is almost only an offset difference. The 3D probe definitely gives better results on that kind of aluminum discs. Because of time constraints, no other discs were analyzed. It may be interesting to check whether the Aluminum0 and Aluminum1 samples are exceptions or not, and what result they would give if they were analyzed by the 3D system. Another point to mention is that the 3D system is very slow compared to the 2D system. The quality enhancement on aluminum discs with the 2D system could be useful to get an overview of the disc, and then decide if a higher quality version of the disc should be taken or not.

Memovox discs

The Memovox samples give very strange spectra. The first sample can be seen in Figure 6-33. The spectrum is difficult to analyze without listening to the sound itself. When listening to the audio file, the optical flow method gives a much better result. These discs seem to be the optimal discs for the optical flow method because they have a lot of edges. On the other hand, the Max derivative method was not at all developed for such discs.

When looking at the figure below, we see that the spectrum found with the Max derivative number 4 is almost constant between 1 and 10kHz. This mean that the noise level is quite high compared to the audio signal. Oppositely, the green spectrum obtained from the optical flow decreases of approximately 8dB per decade in the middle frequencies. It is difficult to verify a difference about the signal level, but the noise level is clearly lower with the optical flow method.

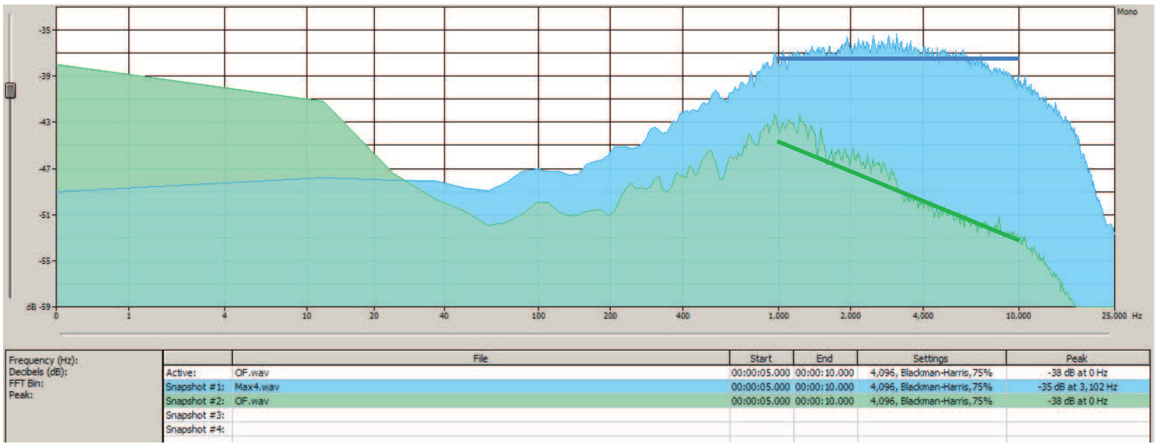


Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OF.wav	00:00:43.099	00:00:43.124	4,096, Blackman-Harris,75%	-39 dB at 1,262 Hz
Snapshot #1:	Max4.wav	00:00:05.000	00:00:10.000	4,096, Blackman-Harris,75%	-40 dB at 2,035 Hz
Snapshot #2:	OF.wav	00:00:05.000	00:00:10.000	4,096, Blackman-Harris,75%	-45 dB at 936 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-33: Memovox0a sample with Max derivative in blue and optical flow in green

On the second sample, a similar frequency response is found, as seen in Figure 6-34. This second sample also seems to have more peaks and valleys in the middle frequencies, which means that the sound should have a higher quality.



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OF.wav	00:00:05.000	00:00:10.000	4,096, Blackman-Harris,75%	-38 dB at 0 Hz
Snapshot #1:	Max4.wav	00:00:05.000	00:00:10.000	4,096, Blackman-Harris,75%	-35 dB at 3,102 Hz
Snapshot #2:	OF.wav	00:00:05.000	00:00:10.000	4,096, Blackman-Harris,75%	-38 dB at 0 Hz
Snapshot #3:					
Snapshot #4:					

Figure 6-34: Memovox0b sample with Max derivative in blue and optical flow in green

Since it is very difficult to analyze the spectra, listening to the sound is the last solution. The audio samples have a good quality, but they are still a little noisy. Adjusting some parameters could probably enhance the quality, but, as those samples were only found close to the end of the project, no extra analysis will be made.

6.6 Summary

6.6.1 Results

After finding the best parameters to utilize with the optical flow methods, we see that there is not much improvement when analyzing shellac discs. The results are very close to each other, but the optical flow processing takes more time than the Max derivative number 4. To get the same audio quality with normal images, it is necessary to smooth the image before processing it. This adds a low-pass filter effect on the sound, which should be avoided to keep every small piece of information in the signal. Using side-lightening images with the optical flow resolves this problem of low-pass filter, but the acquisition of the image needs to be done differently.

Different samples have been tested and the results were slightly different for each of them. It is important to point out that each disc has its own particularities and an algorithm could work better on a disc, but it could be worse on another. The different parameters of the optical flow can also be changed depending on the samples. The tests were done with the values that were found as the ideal values.

After looking at usual samples, the tests on special samples happened to be very interesting. Those special samples could not be study with Max derivative algorithm, but the optical flow algorithm works better. The aluminum discs are the first example. The groove bottom of those discs is not very clear. Therefore the Max derivative algorithm has difficulties to find the edges. After comparing with the 3D system, it appears that the optical flow, even if it gives better results than the 2D system for aluminum discs, cannot reach the quality of the 3D system. On the other hand, scanning a disc with the 3D probe can take up to 13 times longer than the 2D system.

Finally, the Memovox discs were the big surprise in this project. Up to now, it has been very difficult to analyze such records. The new method based on optical flow can now extract a good audio quality from those discs. Even if the algorithm does not work as good as expected on normal records, being able to get a better sound from those Memovox discs is a good improvement.

6.6.2 Optical flow method

In this chapter, different advantages and defaults of the optical flow method are given. The method is not perfect and can probably still be ameliorated. Its biggest advantage is the fact that it can be applied to all kinds of discs. The algorithm is not only written for one type of groove; it can analyze single grooves, multi-grooves from side-lightening, no logical groove as seen on the Memovox (see Figure 6-4) … It can even find audio contents from the groove top in case of the groove bottom is not clear. It becomes some kind of universal method, which provides more or less good audio quality depending on the samples to analyze.

When looking at the signal to noise ratio, it was shown that the noise is usually lower in the higher frequencies. This is due to the fact that the noise present on every velocity vector is averaged. Oppositely, the noise in the middle frequencies tends to be a little higher than with the traditional groove detection methods.

Another point to take into account is that the processing time is a bit less than 2 times longer than with the Max derivative number 4. This can be something we should pay attention when scanning a lot of samples. Another aspect to be careful with is the general volume of the audio. After doing all the tests, the global volume of the sound can change depending on the optical flow parameters. Corrections are done with the default parameters, but changing them may modify the volume of the sound. One last point is that the averaging of the velocity vectors could probably still be perfected. The different methods developed during this project almost all gave the same results. A better management of the velocity vectors themselves when averaging could enhance the quality of the audio signal.

6.6.3 Tracking adjustment

The way which the tracking is done can change a lot the results (see Chapter 3.4.3 for more details about the tracking). The automatic tracking does not work with special samples and it must be set manually. The Fourier method of tracking, which allows the user to change some parameters, can also be used. With this kind of tracking, it is possible that some points of the groove appear out of the tracking. This would generally decrease the quality of the sound.

It is also possible that the amplitude of the groove is really big. This would result in tracking part of the groove top, as it can be seen in Figure 6-35. This often happens with the aluminum samples.

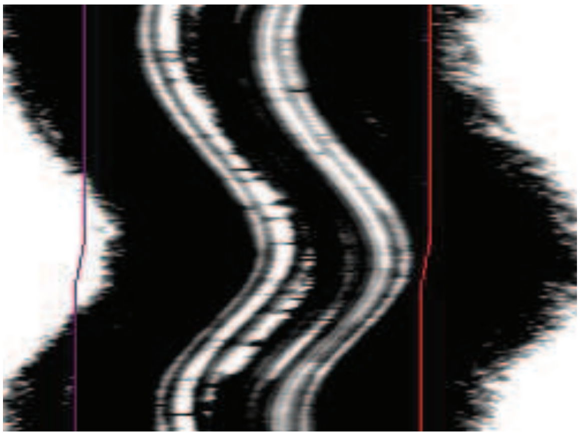


Figure 6-35: The tracking goes over the groove top

One last example is about the Memovox samples. The groove center is very hard to find. The tracking is then more complicated and the results can be worsened if it is not done correctly.

Chapter 7: Improvement on the RENE software

7.1 Groove detection recast

7.1.1 Concept

The groove detection part of the RENE software has been completely modified during this project. This was done to simplify the integration of new detection methods and to lighten the code. Until now, the whole code was written in one main file. The goal of this project on the programming side was to write new code based on object oriented programming. As it would have been very difficult to keep two codes completely different, it was decided to recast the actual groove detection code into a new object oriented structure.

The first point that should be mentioned is that the user interface (GUI) and the algorithmic part should not be written in the same files. The GUI will be separated from the image processing part. The structure of the algorithmic part is described in Chapter 7.1.2 and the GUI in Chapter 7.1.3.

The GUI should show the right parameters depending on which groove detection method is selected. So far, every parameter was written in one interface. This interface was the same for every groove detection method, but not every method was using every parameter. If the user does not know which parameter works with which method, it can be very hard to get a hand on the functionality.

Then, once the user has chosen the groove detection method, the right groove detection class is created depending on the GUI. All the groove detection classes (one for each method) inherit the main “AbstractGrooveDetection” class.

Launching the groove detection method corresponds now to 5 simple calls:

```
// Initialize the groove detection method depending on the GUI
AbstractGrooveDetection grooveDetection = getGrooveDetectionFromGUI(min, max, fit,
kernel);

// Set the different parameters
rooveDetection.setImage(img.height, img.width, b_all);
grooveDetection.setTrack(tr, min, max, trackBin, tr2);
grooveDetection.setForm(this);

// Process the groove detection - All the groove detection methods are called by
this function
grooveDetection.process();
```

The function labeled “getGrooveDetectionFromGUI” returns the right groove detection method depending on the user choice and is explained more in detail in Chapter 7.1.4.

7.1.2 Algorithmic part

As mentioned in the previous chapter, the “AbstractGrooveDetection” class is the base of every groove detection methods. Each new groove detection method should extend this class or one of its children. It was then decided to divide the methods into two categories:

- Methods based on finding the edges of the groove (“AbstractEdgeDetection”)
- Methods based on finding the velocity of the groove (“AbstractFlowMotion”)

The first category (“AbstractEdgeDetection”) includes every existing method developed before this project started. Therefore a complete recast of their structure was needed in order to be able to add them to the wanted structure. A compact class diagram of this structure is shown in Figure 7-1.

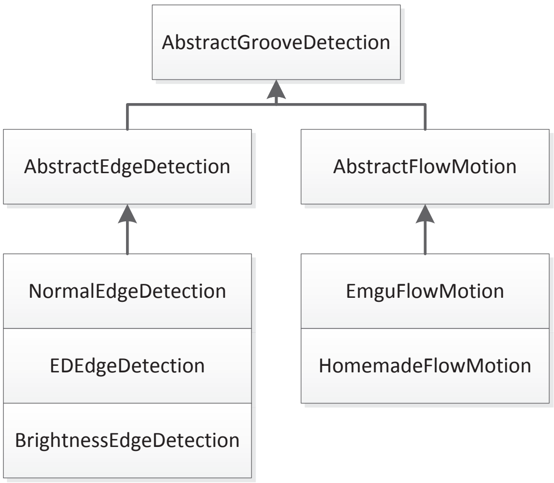


Figure 7-1: Compact class diagram of the different algorithms

The “EmguFlowMotion” class manages every optical flow methods used by the Emgu CV library. The “HomemadeFlowMotion” was developed at the beginning of the project to check how an optical flow algorithm should work. It is essentially a brute-force algorithm which is not very effective.

On the left side of the diagram, the 3 different classes correspond to methods that were already implemented in the RENE software. The functions were not changed, but they were just adapted to match this new structure. The “NormalEdgeDetection” class holds 3 different methods: the Max derivative, the threshold, and the zero crossing methods. These could not be separated in different files, because they interact between each other depending on the parameters.

A complete class diagram can be found in Appendix 10.7.1.

7.1.3 GUI

The user interface has been completely changed. Depending on the groove detection method chosen, it will show a different interface with different parameters. Each interface is written in a new file and one base class manages which interface should be displayed. This base class is called “GrooveDetectionControl” and extends a “UserControl” to be able to plug it into the main program.

The listing below shows the initialization part of this main class. An enumeration lists every different interface and a name is given to each of them.

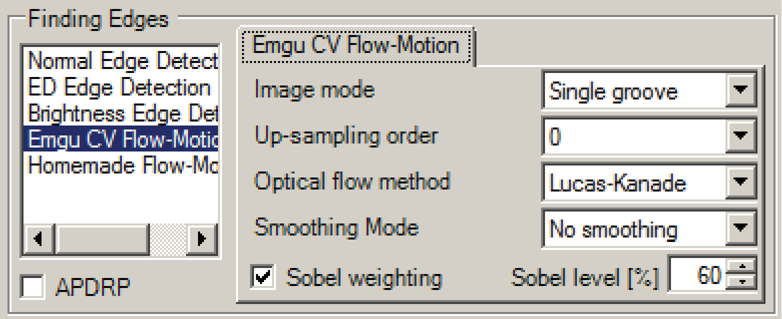
```
public enum GrooveDetectionMethod
{
    NORMAL_EDGE_DETECTION, ED_EDGE_DETECTION, BRIGTHNESS_EDGE_DETECTION,
    EMGU_CV_FLOWMOTION, HOMEMADE_FLOWMOTION
};

private List<string> methodNames = new List<string>() {
    "Normal Edge Detection",
    "ED Edge Detection",
    "Brightness Edge Detection",
    "Emgu CV Flow-Motion",
    "Homemade Flow-Motion",
};
```

As already mentioned, each new groove detection method has its own interface. This interface must extend the class labeled “AbstractMethodGUI”. It can then be simply added to the enumeration and name list. The last part is the initialization of the interfaces themselves. These are declared in a list of "AbstractMethodGUI" and must be declared in the same order of the enumeration.

```
private List<AbstractMethodGUI> methodTabs = new List<AbstractMethodGUI>() {
    new NormalEdgeDetectionGUI(),
    new EDEdgeDetectionGUI(),
    new NoParameterGUI("Audio is contained in the groove brightness instead of the motion."),
    new EmguFlowMotionGUI(),
    new NoParameterGUI("Brute-force optical flow method for finding the groove velocity.")
};
```

Once this is done, the program updates automatically the interface depending on the user choice. In the next figures, we can see different screenshots of the groove detection control.



Finding Edges

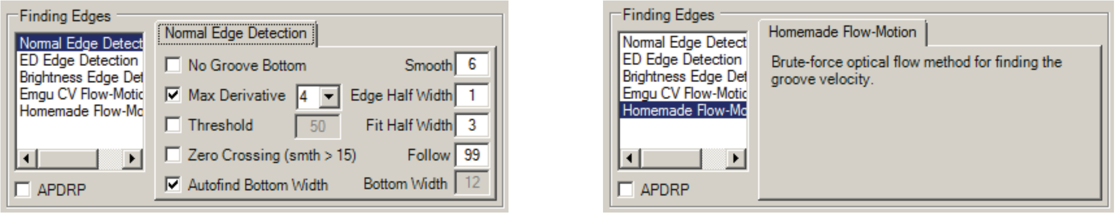
Normal Edge Detect
ED Edge Detection
Brightness Edge Det
Emgu CV Flow-Motic
Homemade Flow-Mc

□ APDRP

Emgu CV Flow-Motion

Image mode: Single groove
Up-sampling order: 0
Optical flow method: Lucas-Kanade
Smoothing Mode: No smoothing
☑ Sobel weighting   Sobel level [%] 60

Figure 7-2: Groove detection user interface



Finding Edges

Normal Edge Detect
ED Edge Detection
Brightness Edge Det
Emgu CV Flow-Motic
Homemade Flow-Mc

□ APDRP

Normal Edge Detection

□ No Groove Bottom   Smooth 6
☑ Max Derivative 4   Edge Half Width 1
□ Threshold 50   Fit Half Width 3
□ Zero Crossing (smth > 15)   Follow 99
☑ Autofind Bottom Width   Bottom Width 12

Finding Edges

Normal Edge Detect
ED Edge Detection
Brightness Edge Det
Emgu CV Flow-Motic
Homemade Flow-Mc

□ APDRP

Homemade Flow-Motion

Brute-force optical flow method for finding the groove velocity.

Figure 7-3: Screens with other selected methods

A complete class diagram can be found in Appendix 10.7.2.

7.1.4 Interface

Creating the algorithm on one side and the GUI on the other side is pretty easy. The complicated part is to connect them together. The algorithm needs to know which parameters the user entered. Another point to take into account is that the user should not be able to modify the entered values when the algorithm is running. There are 2 ways to do that:

- Disabling the user interface
- Make some local copies of the values when the algorithm starts

The second method was kept. The first one is not very user friendly in my opinion, because the GUI would be changing too often. Those values are stored when the constructor of the algorithm is called. In order to call the constructor of the right algorithm depending on the user choice, the function labeled "getGrooveDetectionFromGUI" checks the GUI and returns the right groove detection method. The listing below shows how it works (the parameters passed to the constructors were suppressed for reading purpose).

```
private AbstractGrooveDetection getGrooveDetectionFromGUI(int min, int max, Fit fit,
float[,] kernel)
{
    // Get the method from the GUI
    GrooveDetectionControl.GrooveDetectionMethod method =
        grooveDetectionControl.getGrooveDetectionMethod();

    // Returns the right method with the right parameters
    if (method ==
GrooveDetectionControl.GrooveDetectionMethod.NORMAL_EDGE_DETECTION)
        return new NormalEdgeDetection();

    if (method == GrooveDetectionControl.GrooveDetectionMethod.ED_EDGE_DETECTION)
        return new EDEdgeDetection();

    if (method ==
GrooveDetectionControl.GrooveDetectionMethod.BRIGTHNESS_EDGE_DETECTION)
        return new BrightnessEdgeDetection();

    if (method == GrooveDetectionControl.GrooveDetectionMethod.EMGU_CV_FLOWMOTION)
        return new EmguFlowMotion();

    if (method == GrooveDetectionControl.GrooveDetectionMethod.HOMEMADE_FLOWMOTION)
        return new HomemadeFlowMotion();

    return null;
}
```

On the other hand, the algorithm itself sometimes needs to modify some values of the GUI (for example with the APDRP mode). For this, a set of functions were declared as virtual in the “AbstractMethodGUI”. These functions can be implemented in the children classes if it makes sense to do so. If it is not implemented, the virtual version is automatically called and returns default values.

7.1.5 Documentation

To make the class diagrams, I used a web application called “GenMyModel”. This website provides a free and simple interface to create UML diagrams. An advantage of this is that it can create the documentation in a “.pdf” file from the diagrams if everything is completed correctly. The structure of the file itself cannot be modified, but it gives a good overview of the classes and their purpose.

The file can be found in the folder labeled “SW_version_finale” under the name “Documentation.pdf”. The class diagrams in Appendix can also be found at this link:

- https://repository.genmymodel.com/yann.kurzo/RENE

7.2 Process choice

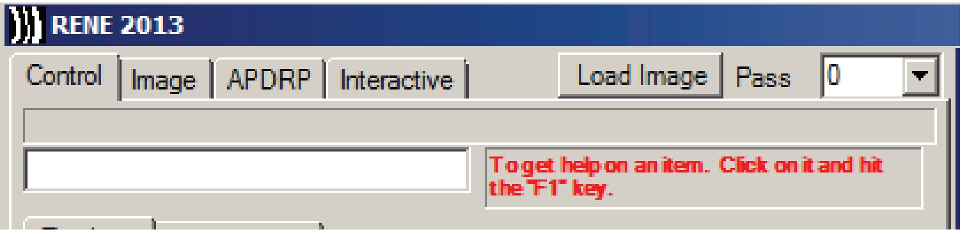
7.2.1 Explanation

The existing RENE program did not allow the user to choose which process he wanted to execute. The different processes were loading the image, tracking, detecting the groove, and sound processing (the details about those processes were already explained in Section 3.4).

Up to now, a single button existed to load the image. When the image was chosen, the whole process was executed. Then, there was no way to apply another groove detection method on the same image without reloading it. This takes some time and it is not very convenient when testing new algorithms.

It was chosen to modify the user interface to allow the user to choose between the processes. It should be for example possible for him to only select another groove detection method without reloading and tracking the image. Changing the output filter applied during the sound processing phase should be possible without searching the groove again.

These modifications would considerably enhance the testing capabilities of the program.



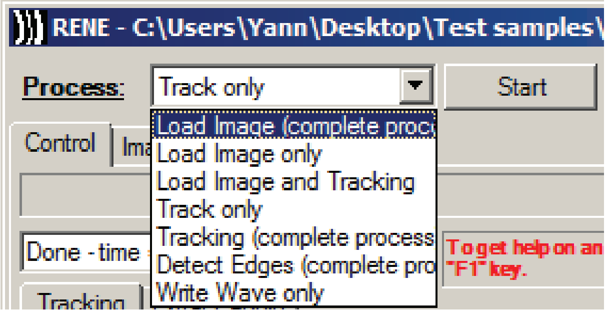
To get help on an item. Click on it and hit the "F1" key.

Figure 7-4: Old user interface

7.2.2 Modifications

As shown in Figure 7-5, we have a certain number of processes that can be chosen. Some processes cannot be used if the previous step in the program has not been executed. To be able to check which process is allowed to be executed, I decided to assign to each process a certain level of execution. Those levels correspond to the different steps of the program which are described in Chapter 3.4.1:

- Loading (level 0)
- Tracking (level 1)
- Groove detection (level 2)
- Sound processing (level 3)


RENE - C:\Users\Yann\Desktop\Test samples\
Start
Load Image (complete proc
Load Image only
Load Image and Tracking
Track only
Tracking (complete process
Detect Edges (complete pro
Write Wave only
Control
Ima
Tracking

Process: Track only

Done - time

To get help on an "F1" key.

Figure 7-5: New user interface with process choices

The following listing shows the different enumeration and variables used to determine the processes level:

```
// Name of the different process choices
private enum ChosenProcess { NO_PROCESS = -1, LOAD_IMAGE, LOAD_ONLY, LOAD_AND_TRACK,
TRACK_ONLY, TRACKING, EDGE_DETECTION, WRITE_WAVE };

// 4 level of process
private enum ProcessLevel { LOAD_LEVEL, TRACK_LEVEL, EDGE_LEVEL, WRITE_LEVEL };

// Level of the different processes
private ProcessLevel[] processLevel = {
        ProcessLevel.LOAD_LEVEL, ProcessLevel.LOAD_LEVEL, ProcessLevel.LOAD_LEVEL,
        ProcessLevel.TRACK_LEVEL, ProcessLevel.TRACK_LEVEL,
        ProcessLevel.EDGE_LEVEL,
        ProcessLevel.WRITE_LEVEL };
```

```
// Resulting level of the different processes
private ProcessLevel[] resultingLevel = {
        ProcessLevel.WRITE_LEVEL,
        ProcessLevel.LOAD_LEVEL,
        ProcessLevel.TRACK_LEVEL, ProcessLevel.TRACK_LEVEL,
        ProcessLevel.WRITE_LEVEL, ProcessLevel.WRITE_LEVEL, ProcessLevel.WRITE_LEVEL
};

// Actual level set to -1
private static int processExecuted = (int)ChosenProcess.NO_PROCESS;
```

The “processLevel” array gives the level of each process and the “resultingLevel” array gives the resulting level after the process is executed. By checking the actual level and the chosen level from the combobox, we can determine whether it can be executed or not.

```
if(processExecuted + 1 >=
(int)processLevel[this.actualProcessComboBox.SelectedIndex])
{
    // Execute process
}
```

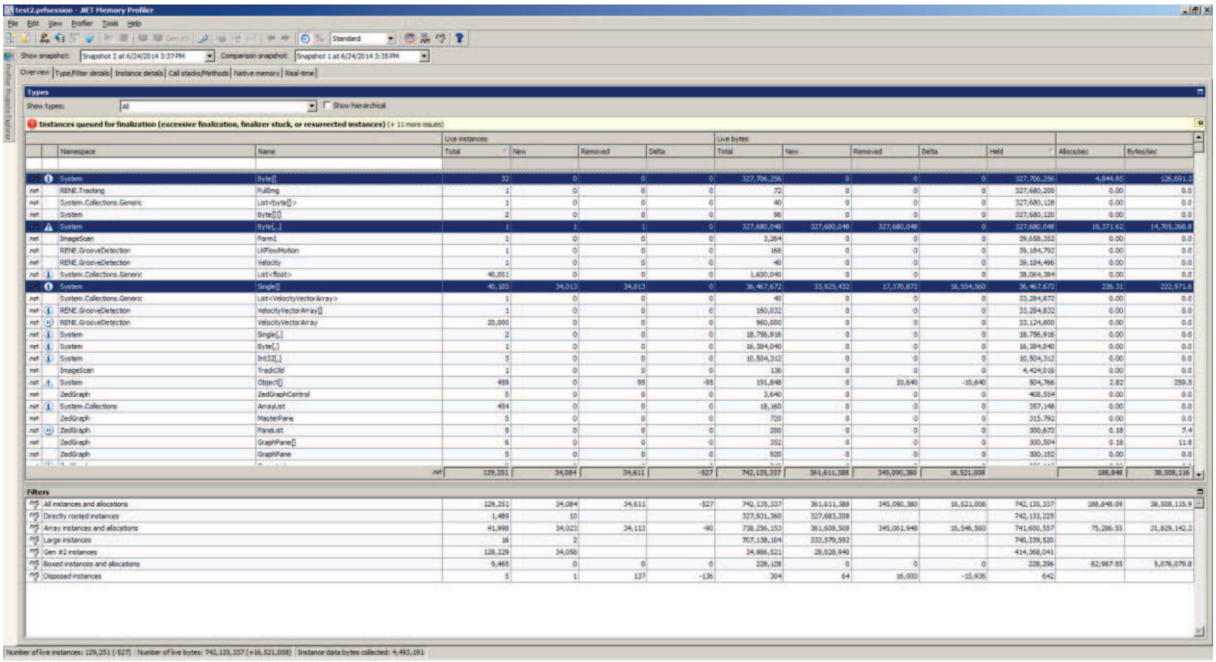
7.3 Optimizations

7.3.1 Memory leaks

At a point during the project, using the optical flow in the RENE software was taking too much time (more than 5 minutes) and using up to 4GB of RAM. This was mostly due to memory leaks. The C# language works with a garbage collector, so it is not easy to find out where the memory is not freed. It seems that it loses tracks on some objects when performing the optical flow algorithm.

A lot of “List<>” objects which are resizable arrays are used in the program. In my case, the number of elements can get very high (it depends on the size of the images) and reallocating items takes time. If the program needs to reallocate memory, it basically creates a new place in memory and copies the existing values into it. This takes time and this is why allocating a defined array size at the beginning of the program execution increases the processing speed. I also started to use the “dispose()” method on the pictures to tell the garbage collector that they can be freed.

It already helped but the memory usage was still at 2GB. I finally used the .Net Memory Profiler tool to find out where the memory leaks come from. As explain in Chapter 4.1.3, it takes snapshots of the memory at different timestamps and then analyses the differences. A screenshot of the analysis can be seen in Figure 7-6.


test2.prfsession - .NET Memory Profiler
File Edit View Profiler Tools Help
Standard
Show snapshot: Snapshot 2 at 6/24/2014 3:37 PM
Comparison snapshot: Snapshot 1 at 6/24/2014 3:35 PM
Overview | Type/Filter details | Instance details | Call stacks/Methods | Native memory | Real-time
Profiler Projects Explorer
Types
Show types: All
Show hierarchical

Instances queued for finalization (excessive finalization, finalizer stuck, or resurrected instances) (+ 11 more issues)

		Live instances				Live bytes						
Namespace	Name	Total	New	Removed	Delta	Total	New	Removed	Delta	Held	Allocs/sec	Bytes/sec
System	Byte[]	32	0	0	0	327,706,256	0	0	0	327,706,256	4,844.85	126,691.2
RENE.Tracking	FullImg	1	0	0	0	72	0	0	0	327,680,200	0.00	0.0
System.Collections.Generic	List<byte[]>	1	0	0	0	40	0	0	0	327,680,128	0.00	0.0
System	Byte[][]	2	0	0	0	96	0	0	0	327,680,120	0.00	0.0
System	Byte[,,]	1	1	1	0	327,680,048	327,680,048	327,680,048	0	327,680,048	19,371.62	14,705,268.8
ImageScan	Form1	1	0	0	0	3,264	0	0	0	39,658,352	0.00	0.0
RENE.GrooveDetection	LKFlowMotion	1	0	0	0	168	0	0	0	39,184,792	0.00	0.0
RENE.GrooveDetection	Velocity	1	0	0	0	40	0	0	0	39,184,496	0.00	0.0
System.Collections.Generic	List<float>	40,001	0	0	0	1,600,040	0	0	0	38,064,384	0.00	0.0
System	Single[]	40,105	34,013	34,013	0	36,467,672	33,925,432	17,370,872	16,554,560	36,467,672	226.31	222,971.6
System.Collections.Generic	List<VelocityVectorArray>	1	0	0	0	40	0	0	0	33,284,872	0.00	0.0
RENE.GrooveDetection	VelocityVectorArray[]	1	0	0	0	160,032	0	0	0	33,284,832	0.00	0.0
RENE.GrooveDetection	VelocityVectorArray	20,000	0	0	0	960,000	0	0	0	33,124,800	0.00	0.0
System	Single[,]	2	0	0	0	18,756,916	0	0	0	18,756,916	0.00	0.0
System	Byte[,]	1	0	0	0	16,384,040	0	0	0	16,384,040	0.00	0.0
System	Int32[,]	3	0	0	0	10,504,312	0	0	0	10,504,312	0.00	0.0
ImageScan	TrackOld	1	0	0	0	136	0	0	0	4,424,016	0.00	0.0
System	Object[]	459	0	95	-95	191,848	0	10,640	-10,640	504,766	2.82	259.5
ZedGraph	ZedGraphControl	5	0	0	0	3,640	0	0	0	408,554	0.00	0.0
System.Collections	ArrayList	454	0	0	0	18,160	0	0	0	357,148	0.00	0.0
ZedGraph	MasterPane	5	0	0	0	720	0	0	0	315,792	0.00	0.0
ZedGraph	PaneList	5	0	0	0	200	0	0	0	300,672	0.18	7.4
ZedGraph	GraphPane[]	6	0	0	0	352	0	0	0	300,504	0.18	11.8
ZedGraph	GraphPane	5	0	0	0	920	0	0	0	300,152	0.00	0.0
	.net	129,251	34,084	34,611	-527	742,135,337	361,611,388	345,090,380	16,521,008		188,848	38,508,116

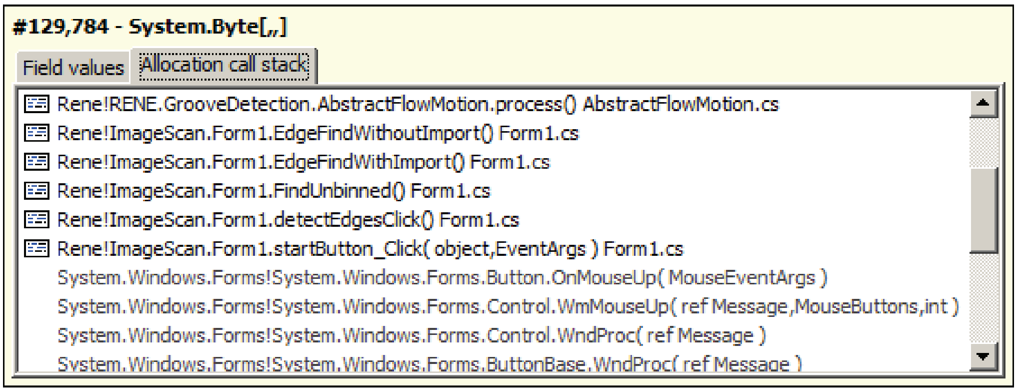
Filters

All instances and allocations	129,251	34,084	34,611	-527	742,135,337	361,611,388	345,090,380	16,521,008	742,135,337	188,848.09	38,508,115.9
Directly rooted instances	1,489	10			327,931,360	327,683,208			742,132,225		
Array instances and allocations	41,998	34,023	34,113	-90	738,256,153	361,608,508	345,061,948	16,546,560	741,600,557	75,286.55	31,829,142.2
Large instances	16	2			707,138,104	333,579,592			740,339,520		
Gen #2 instances	128,229	34,058			34,986,521	28,028,940			414,368,041		
Boxed instances and allocations	9,465	0	0	0	228,128	0	0	0	228,296	62,987.55	5,076,079.8
Disposed instances	5	1	137	-136	304	64	16,000	-15,936	642		

Number of live instances: 129,251 (-527) | Number of live bytes: 742,135,337 (+16,521,008) | Instance data bytes collected: 4,493,191

Figure 7-6: Memory comparison at 2 different timestamps in the RENE program

The tool tells us that there are memory problems at different points in the program. The 3 points in blue in the previous figure show some variables that use a lot of memory for nothing. If we select one of points, we get the figure below. It tells us from which function the problem comes so that it is possible to find the memory leak cause.



#129,784 - System.Byte[,,]

```
Rene!RENE.GrooveDetection.AbstractFlowMotion.process() AbstractFlowMotion.cs
Rene!ImageScan.Form1.EdgeFindWithoutImport() Form1.cs
Rene!ImageScan.Form1.EdgeFindWithImport() Form1.cs
Rene!ImageScan.Form1.FindUnbinned() Form1.cs
Rene!ImageScan.Form1.detectEdgesClick() Form1.cs
Rene!ImageScan.Form1.startButton_Click( object,EventArgs ) Form1.cs
System.Windows.Forms!System.Windows.Forms.Button.OnMouseUp( MouseEventArgs )
System.Windows.Forms!System.Windows.Forms.Control.WmMouseUp( ref Message,MouseButtons,int )
System.Windows.Forms!System.Windows.Forms.Control.WndProc( ref Message )
System.Windows.Forms!System.Windows.Forms.ButtonBase.WndProc( ref Message )
```

Figure 7-7: Zoom on one memory problem

With that tool, I finally found the main spots which created those memory leaks. Here is an example:

```
Image<Gray, float> img = slice.Sobel(1, 0, 3).Convert<Gray, byte>();
```

The “sobel” function creates a temporary picture. This picture is then converted and stored in the “img” variable. The temporary created picture is then lost and the garbage collector cannot free it. This line was executed for each slice of the whole image, so that the memory was increasing very fast. These extra lines solved the problem:

```
Image<Gray, float> sobeledSlice = slice.Sobel(1, 0, 3);
Image<Gray, byte> img = sobeledSlice.Convert<Gray, byte>();
sobeledSlice.Dispose();
```

The actual program uses about 800MB of RAM, and has a spike to approximately 1.1GB when converting the image from the raw data to the EMGU image, as we can see in Figure 7-8.

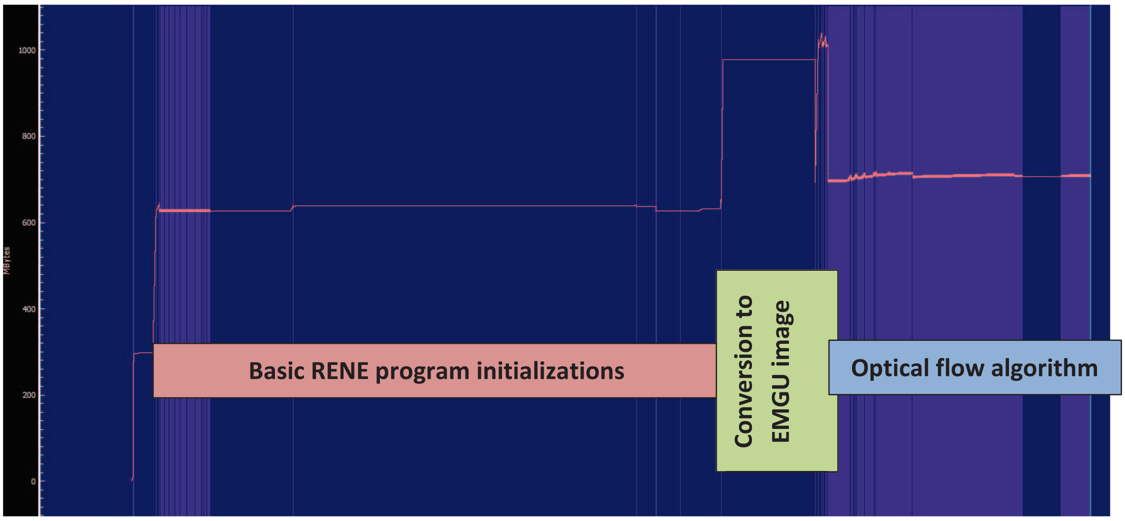


Figure 7-8: Memory usage of RENE when using optical flow methods

7.3.2 Speed

To improve the execution speed, I used some little tricks. The first one is to decrement the indexes when using “for” loops instead of incrementing them. The assembly instruction to compare to zero is much faster than comparing other constant values.

```
for (int w = width - 1; w >= 0; w--)

Instead of

for (int w = 0; w < width; w++)
```

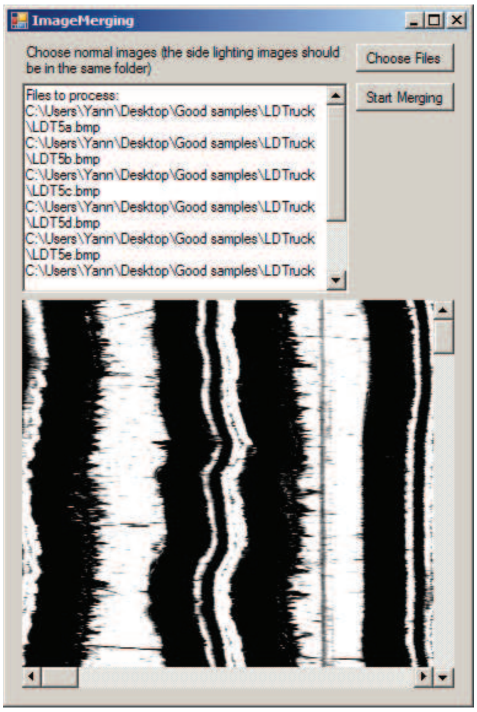
It seems that using C# properties within a loop is not very efficient [27]. I tried to avoid using them. I also tried to avoid copying data (mostly pictures) every time it is not needed. For EMGU pictures, it is also interesting to use the ROI (Region of Interest) as much as possible. It defines a rectangle on the picture, which delimits the border of the picture itself. After that, every algorithm performed on the picture will only be done on the ROI defined by the rectangle.

I decreased the algorithm running time from about 5 minutes to 90 seconds with the memory leaks problems. After that, changing the loops and the copy of pictures decreased the time to approximately 45 seconds, which is much better. In comparison, the max derivative edge detection method takes about 25 seconds. As we are doing a lot of image processing, it is almost impossible to decrease this time more.

The actual execution time of the algorithm depends on different parameters. It is mainly determined by the size of the image, but also by the groove size defined during the tracking, the up-sampling order, or even the smoothing method. It can go from around 30 seconds to 90 seconds.

7.4 Image merging software

A simple program was also made to merge images with only one groove and side-lightening images together. It was developed to save time when modifying the side-lightening images. Finally this program was not used a lot because of the superposition problem described in Chapter 5.4.2.



Choose normal images (the side lighting images should be in the same folder)

```
Files to process:
C:\Users\Yann\Desktop\Good samples\LDTruck
\LDT5a.bmp
C:\Users\Yann\Desktop\Good samples\LDTruck
\LDT5b.bmp
C:\Users\Yann\Desktop\Good samples\LDTruck
\LDT5c.bmp
C:\Users\Yann\Desktop\Good samples\LDTruck
\LDT5d.bmp
C:\Users\Yann\Desktop\Good samples\LDTruck
\LDT5e.bmp
C:\Users\Yann\Desktop\Good samples\LDTruck
```

Figure 7-9: Image merging software screenshot

Chapter 8: Conclusion

8.1 Summary and discussion of the work performed

The optical flow algorithm developed during this project is working and could open new ways of analyzing old audio records. After testing the algorithm with a lot of different parameters which did not modify the results a lot, the default parameters for getting good results were chosen. Two different ways of processing the image gave the best results. The first one works with applying a smoothing filter before utilizing the optical flow algorithm. This increases the audio quality, but on the other hand it creates a low-pass filter effect on the sound. This effect should be avoided when reconstructing very old samples from libraries or archives for example. They usually prefer to get a lower quality with every little piece of information than a higher quality which risks deleting some of the signal information.

The idea of using side-lightening images is not new, but the optical flow algorithm is the first one that can handle them quite correctly. Those images generally have more information because they are constructed with more than one groove. By analyzing those images with the optical flow, the noise contained on the edges of each groove is averaged, enhancing the sound quality.

It was also found that the results of the algorithms really depend on the types of samples. Some samples can give better results with the optical flow, but some others could be better with the traditional methods. This is true for shellac discs, but other special samples gave surprising results. The aluminum discs are usually hard to study with the traditional methods, but the quality gets better with the optical flow. However those discs are generally acquired by the 3D probe which still gives a better result than the new algorithm.

The best results obtained so far were with the Memovox samples. The optical flow algorithm is the first one to get that kind of results with those records. The Memovox discs have a high historical importance. The few samples that were studied were for example recorded during the WWII. The Library of Congress in Washington D.C. possesses some discs and it would be very interesting to study them [28].

On the programming side of the project, the recast of the groove detection part of the program can be considered as successful. The new structure is completely object oriented and adding new groove detection methods from now on is way less complicated than it used to be. It was not an easy task to modify the program due to its structure. The final version has a GUI completely separated from the algorithmic part. The result is a flexible user interface to which new methods can be straightforwardly added.

8.2 Further work

The tests done on special samples were started quite late in the project. The Memovox discs are quite unusual and I knew their existence only by the end of the project. Some more tests on those samples would be needed. The optical flow algorithm could also be adapted to give better results on this kind of records.

The side-lightening images give quite good results and the noise is minimized. It could be interesting to modify the acquisition system to get more than 2 or 3 grooves. An idea would be to get information from the groove sides with different lightening angles. Theoretically the more edges there are, the better the optical flow algorithm should react.

About the programming, the complete optical flow process is certainly not perfect. This was a brand new topic for everybody and ameliorations can still probably be done. It could be interesting to use different optical flow algorithms (other than Lucas-Kanade or Horn-Schunck for example) to check for enhancement. Different image processing libraries may be used to analyze the images otherwise. The algorithm which detects the good velocity vectors from the bad ones could be ameliorated by taking into consideration the previous and next values during the averaging process. An ideal case would be to study each edge separately and determine if the groove is in good shape enough to be taken into consideration. This can be very difficult to implement because of the vast quantity of different types of samples.

Another way that could be developed for detecting the groove is by using machine learning algorithms. The software could “learn” to read discs from known samples. It could then try to apply the same method to new samples.

8.3 Personal conclusion

This project offered me the opportunity to study historical records. The Memovox discs were recorded during the Second World War for example. Some other samples were even older. Being able to hear people talking or to listen to music from that far in time is very interesting. The audio reconstruction by image processing is an important tool for historians.

On my side, it gave me new knowledge about image processing. This vast topic is very interesting. The optical flow subject was completely new and opened new ways of analyzing pictures. I had never heard of it before, and that was motivating for me to learn a completely new subject.

The C# programming language was also new for me. Since it is very close to Java, it did not take me too long to get used to it. On the other hand, the structure of the RENE program took me a long time to understand because of its lack of organization. Getting to know even the basic functionalities quickly becomes hard. Also I would say that it was a lot of programming for an electrical engineering bachelor thesis.

Doing my bachelor thesis in Berkeley also gave me the chance of discovering new places, living with a different culture, and meeting new friends. I will keep a nice souvenir from the time I spent in Berkeley. Finally, these three months spent on my bachelor thesis allowed me to improve my English skills. Knowing other languages is very important nowadays and I am very grateful to the people who made this Bachelor thesis feasible.

Chapter 9: Bibliography

[1] Carl Haber, "Imaging Lost Voices Optical Scanning Applied to Recorded Sound Preservation and Access", Lawrence Berkeley National Lab, mars 2009.

[2] "The Phonautograph and Precursors to Edison's Phonograph," Donald C. Davidson Library, University of California, Santa Barbara, [Online]. Available: http://cylinders.library.ucsb.edu/history-early.php. [Accessed in July 2014].

[3] The Associated Press, "1878 Edison audio unveiled: Tinfoil-phonograph recording provides soundtrack to history," 2012. [Online]. Available: http://www.oregonlive.com/music/index.ssf/2012/10/1878_edison_audio_revealed_tin.html. [Accessed in July 2014].

[4] "Phonograph cylinder," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Phonograph_cylinder. [Accessed in July 2014].

[5] "History of sound recording," [Online]. Available: http://en.wikipedia.org/wiki/History_of_sound_recording#Digital_Recording. [Accessed in July 2014].

[6] William A. Penn and Martha J. Hanson, "Development of a non destructive playback system for cylinder recordings," [Online]. Available: http://www.archeophone.org/brol/doc_externe_a%20_preserver/syracuse_radius_project/The%20Syracuse%20University%20Library%20Radius%20Project.htm. [Accessed in July 2014].

[7] "Need leaned record," [Online]. Available: http://s1075.photobucket.com/user/ggergm/media/photos/needleandrecord8x10.jpg.html. [Accessed in July 2014].

[8] "Gramophone record," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Gramophone_record. [Accessed in July 2014].

[9] "Acetate disc," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Acetate_disc. [Accessed in July 2014].

[10] "Aluminium disc," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Aluminum_disc. [Accessed in July 2014].

[11] Jérémy Singy, "Cracked records with 3D/IRENE", Master Thesis at the University of Applied Sciences Western Switzerland, February 2013.

[12] "C Sharp (programming language)," [Online]. Available: http://en.wikipedia.org/wiki/C_Sharp_(programming_language). [Accessed in July 2014].

[13] "EMGU CV," [Online]. Available: http://www.emgu.com/wiki/index.php/Main_Page. [Accessed in July 2014].

[14] "Emgu.CV Namespace," [Online]. Available: http://www.emgu.com/wiki/files/2.4.2/document/Index.html. [Accessed in July 2014].

[15] ".NET Memory Profiler," [Online]. Available: http://memprofiler.com/. [Accessed in June 2014].

[16] "Optical Flow," [Online]. Available: http://en.wikipedia.org/wiki/Optical_flow. [Accessed in May 2014].

[17] Guido Gerig, "Optical Flow I", Part of the "3D Computer Vision" Course, CS6320 Spring 2012, University of Utah, USA.

[18] Marc Pollefeys and Markus Gross, "Optical Flow", part of the Visual Computing course at the ETH, Zürich, Switzerland.

[19] Paul Sastrasinh, "Dense Realtime Optical Flow on the GPU," [Online]. Available: http://cs.brown.edu/courses/csci1290/2011/results/final/psastras/. [Accessed in June 2014].

[20] J. L. Barron and N. A. Thacker, "Tutorial: Computing 2D and 3D Optical Flow", Tina Memo No. 2004-012, January 2005.

[21] Blake Randolph and Robert Sekuler , "The Aperture Problem," 2006. [Online]. Available: http://stoomey.wordpress.com/2008/04/18/20/. [Accessed in July 2014].

[22] Vilayanur S. Ramachandran and Diane Rogers-Ramachandran, "Illusions: Sliding Stripes", Scientific American Mind (June/July 2008), 19, 18-20. [Online]. Available: http://www.nature.com/scientificamericanmind/journal/v19/n3/full/scientificamericanmind0608-18.html. [Accessed in July 2014].

[23] Shahriar Negahdaripour and Chih-Ho Yi, "A Generalized Brightness Change Model for Computing Optical Flow", Proc. on the IEEE Fourth International Conference on Computer Vision (ICCV), Berlin, Germany, May 1993.

[24] Yeon-Ho Kim, Aleix M. Martinez and Avi C. Kak, "A Local Approach for Robust Optical Flow Estimation under Varying Illumination", Proc. on the British Machine Vision Conference (BMCV), Kingston University, London, England, September 2004.

[25] Horst W. Haussecker and David J. Fleet, "Computing Optical Flow with Physical Models of Brightness Variation", Proc. on the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Hilton Head Island, SC / USA, June 2000.

[26] Frank Hoffmann, Encyclopedia of Recorded Sound, New York, NY / USA: Routledge, 2005.

[27] Luca Del Tongo, "Iterate over pixels of an image with emgu cv," 2012. [Online]. Available: http://stackoverflow.com/questions/5101986/iterate-over-pixels-of-an-image-with-emgu-cv. [Accessed in June 2014].

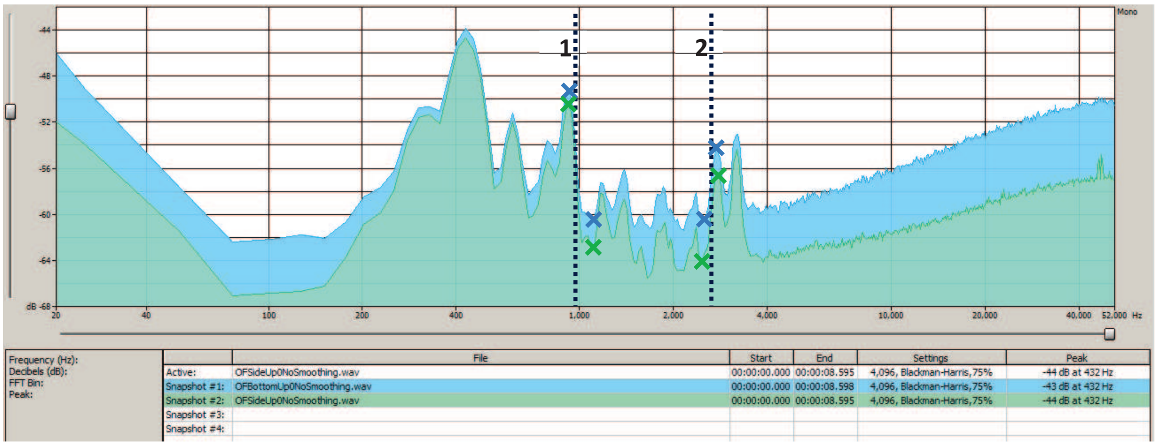
[28] "Library of Congress: Collection Highlights," 2012. [Online]. Available: http://www.loc.gov/rr/record/rssfacts.html. [Accessed in July 2014].

Chapter 10: Appendices

10.1 Normal and side-lightening samples

Normal images in blue and side-lightening in green

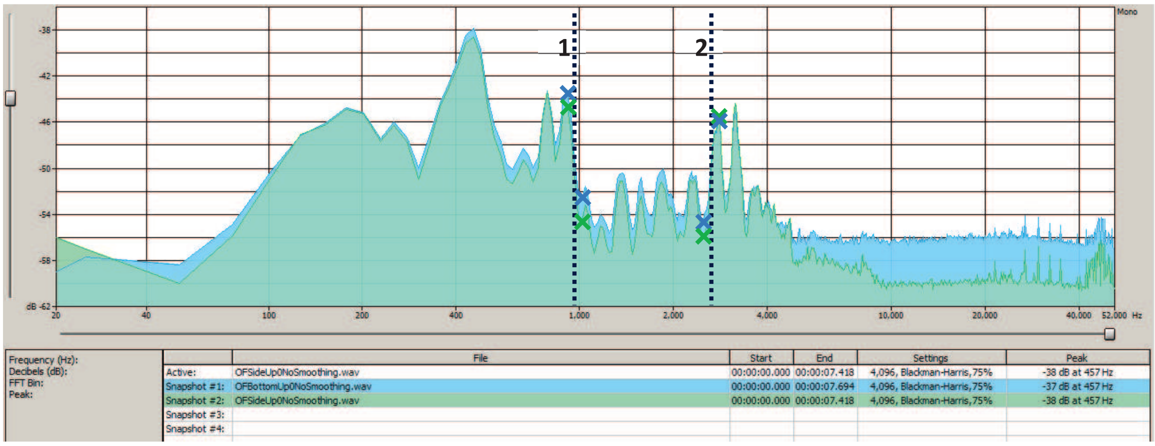
10.1.1 Nobody16



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris, 75%	-43 dB at 432 Hz
Snapshot #2:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #3:					
Snapshot #4:					

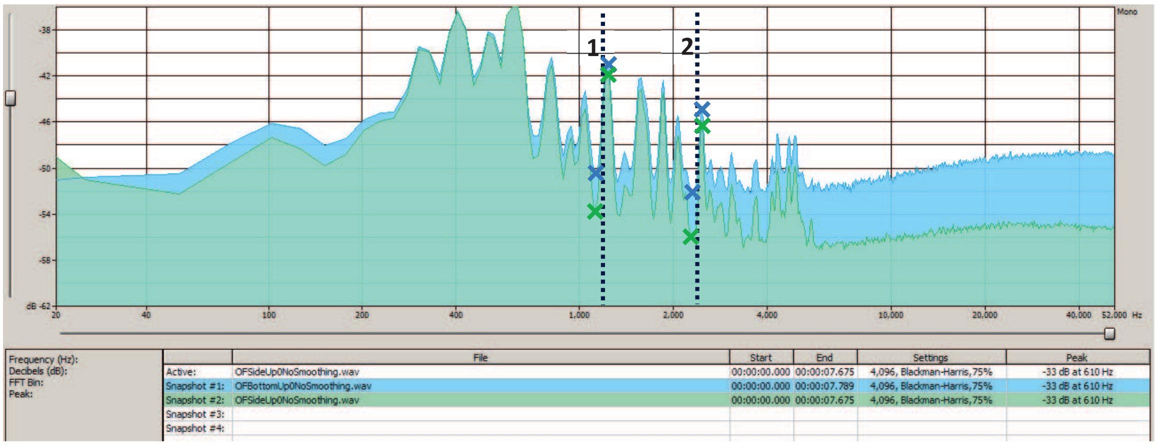
10.1.2 SSB25



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris, 75%	-38 dB at 457 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris, 75%	-37 dB at 457 Hz
Snapshot #2:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris, 75%	-38 dB at 457 Hz
Snapshot #3:					
Snapshot #4:					

10.1.3 DC1



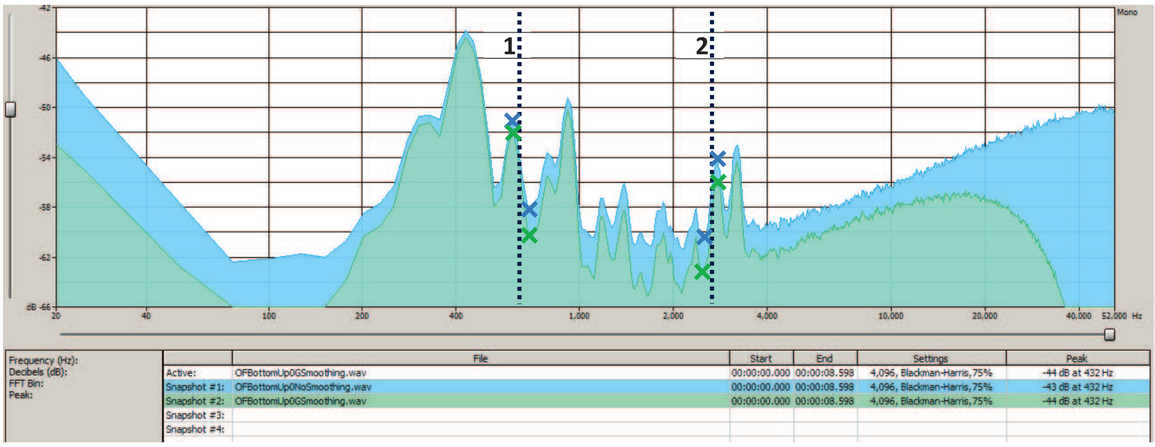
Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #2:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

10.2 Without smoothing and Gaussian filter with normal images

Without smoothing in blue and with Gaussian filter in green

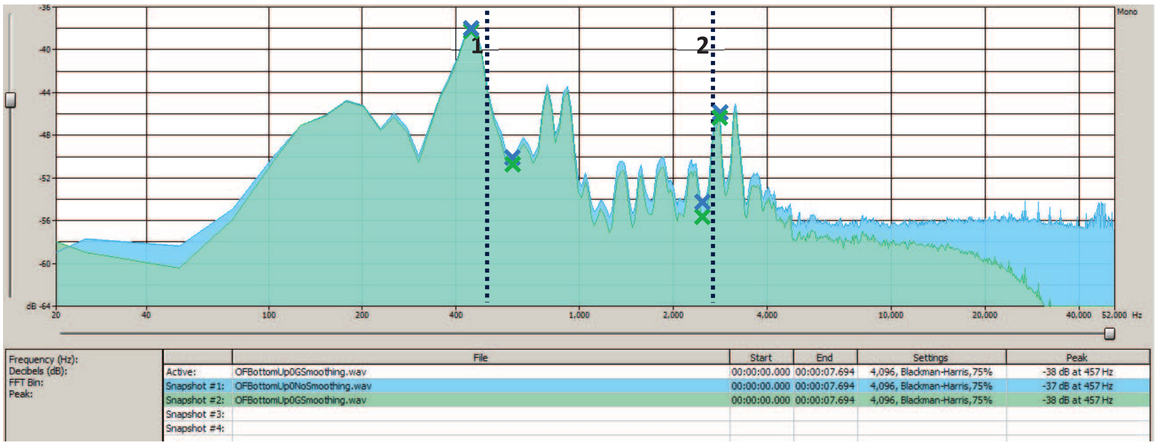
10.2.1 Nobody16



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris,75%	-43 dB at 432 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #3:					
Snapshot #4:					

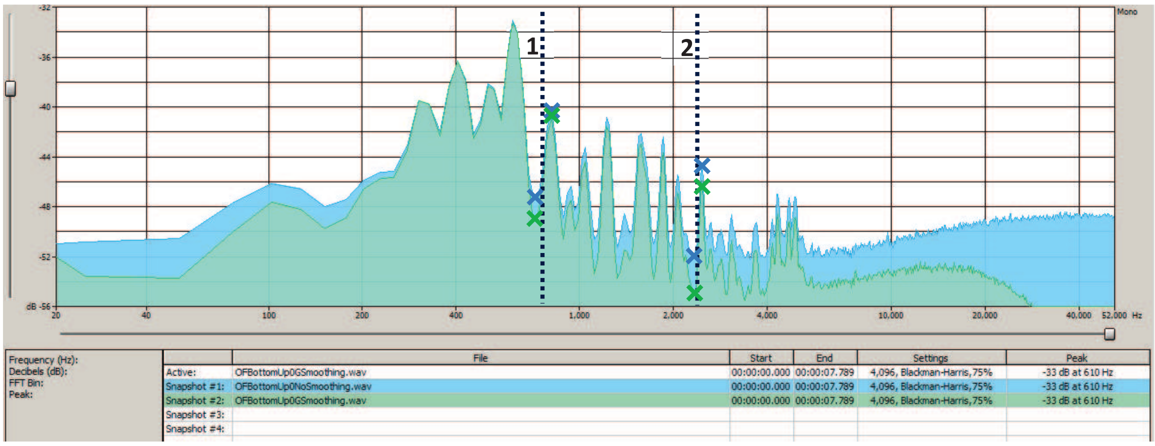
10.2.2 SSB25



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-37 dB at 457 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #3:					
Snapshot #4:					

10.2.3 DC1



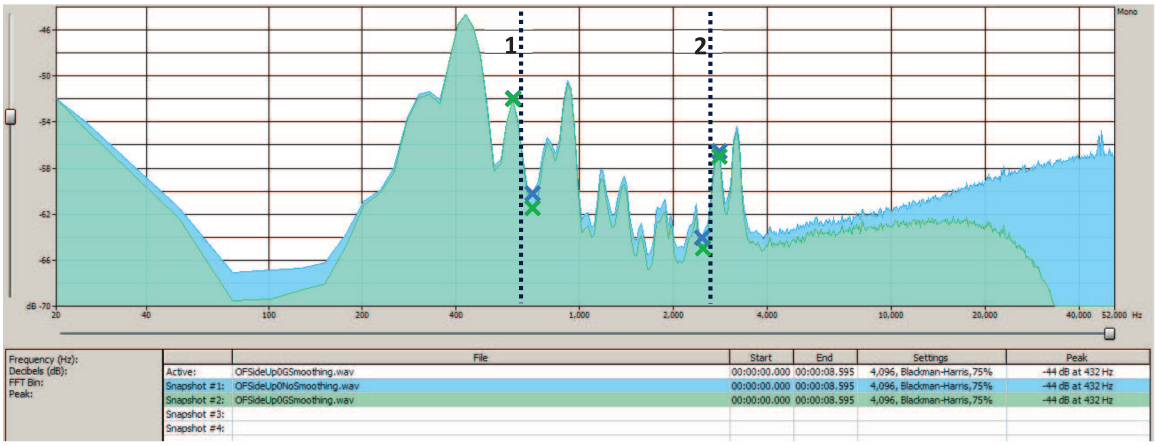
Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

10.3 Without smoothing and Gaussian filter with side-lightening images

Without smoothing in blue and with Gaussian filter in green

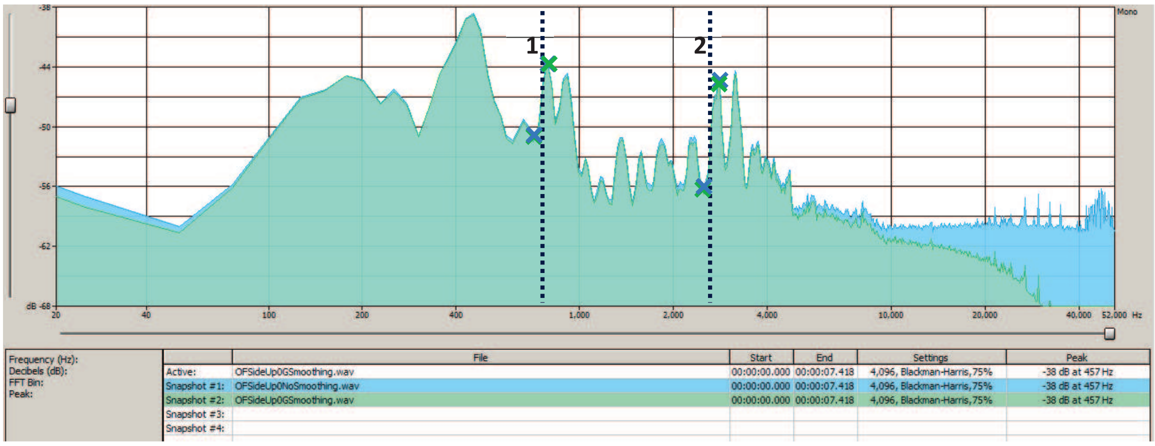
10.3.1 Nobody16



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #2:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #3:					
Snapshot #4:					

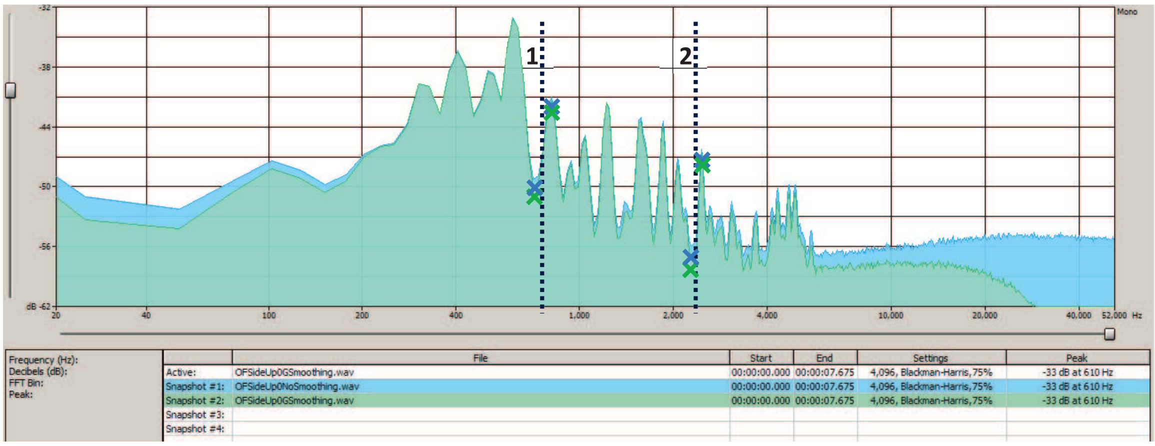
10.3.2 SSB25



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris, 75%	-38 dB at 457 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris, 75%	-38 dB at 457 Hz
Snapshot #2:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris, 75%	-38 dB at 457 Hz
Snapshot #3:					
Snapshot #4:					

10.3.3 DC1



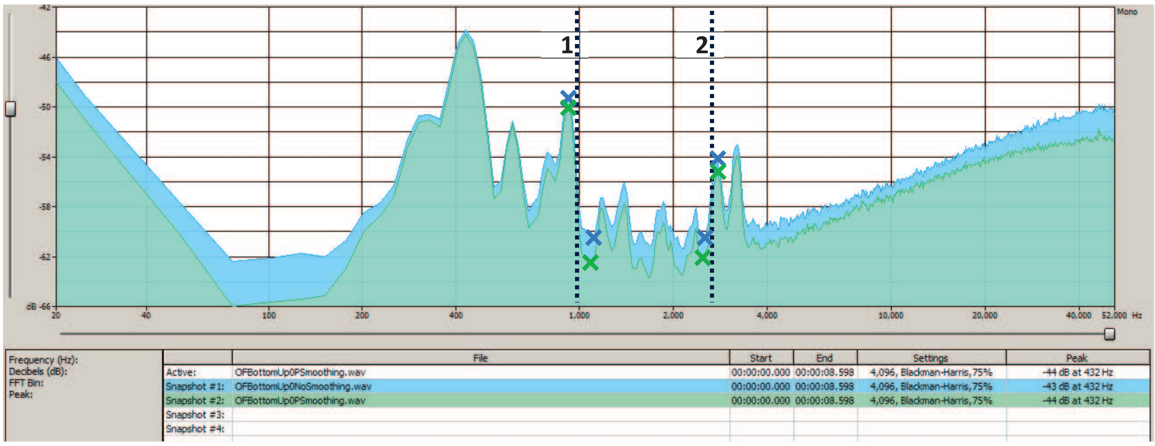
Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #2:	OFSideUp0GSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

10.4 Partial smoothing on normal images

Without smoothing in blue and with partial smoothing in green

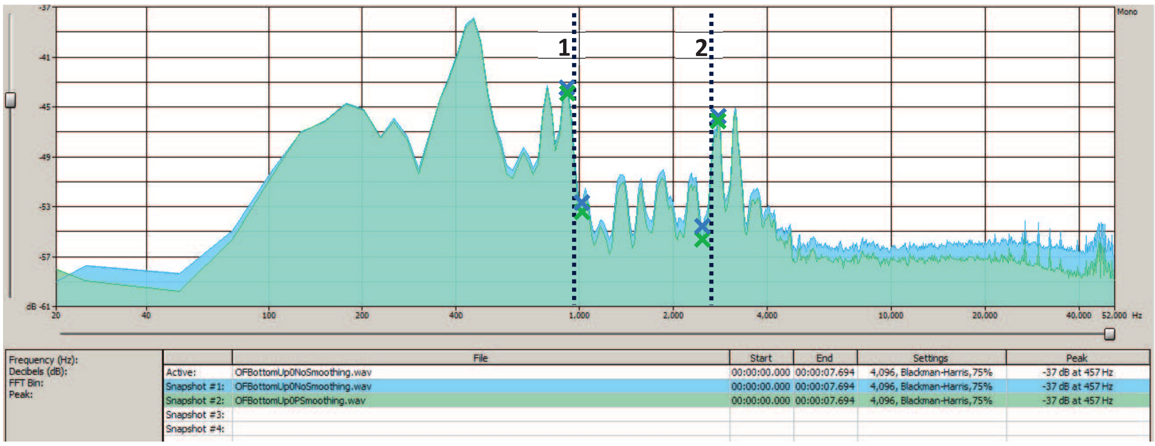
10.4.1 Nobody16



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0PSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris, 75%	-43 dB at 432 Hz
Snapshot #2:	OFBottomUp0PSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris, 75%	-44 dB at 432 Hz
Snapshot #3:					
Snapshot #4:					

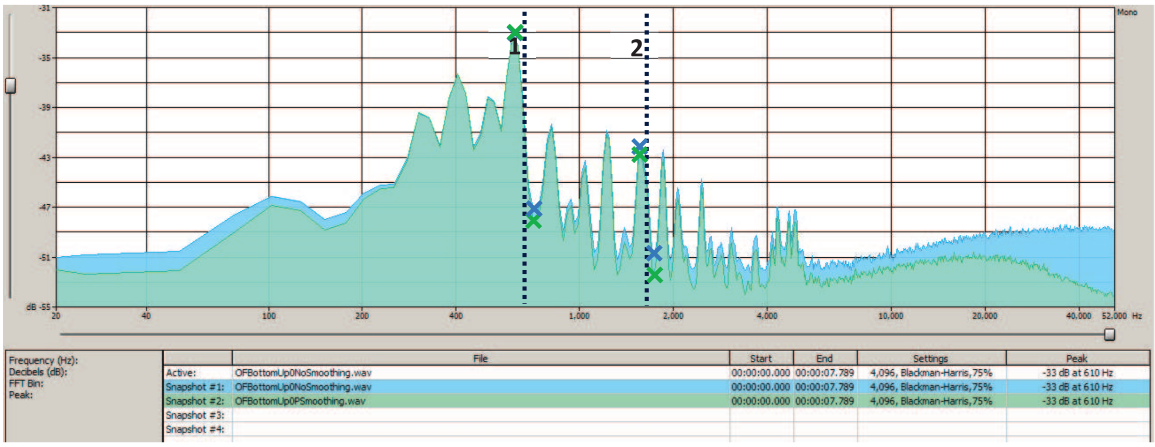
10.4.2 SSB25



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris, 75%	-37 dB at 457 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris, 75%	-37 dB at 457 Hz
Snapshot #2:	OFBottomUp0PSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris, 75%	-37 dB at 457 Hz
Snapshot #3:					
Snapshot #4:					

10.4.3 DC1



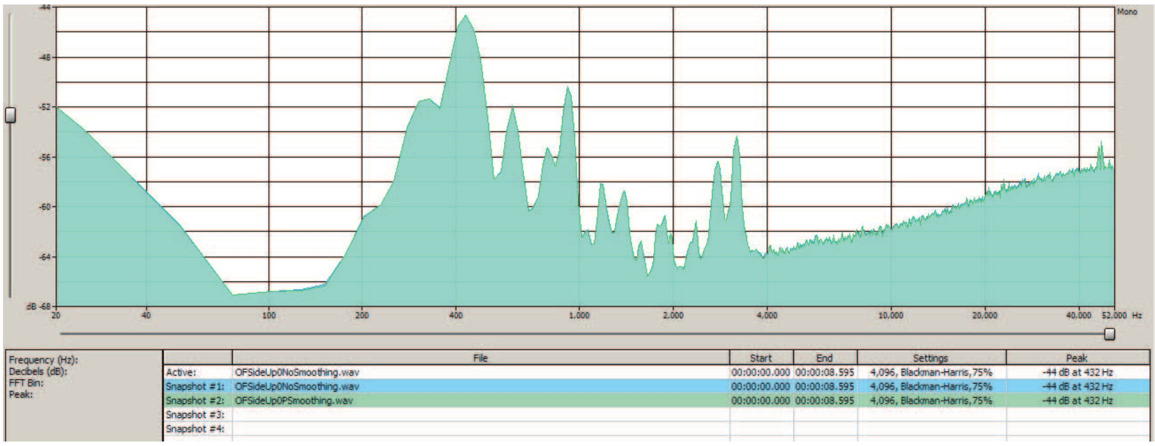
Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #1:	OFBottomUp0NoSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #2:	OFBottomUp0PSmoothing.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris, 75%	-33 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

10.5 Partial smoothing on side-lightening images

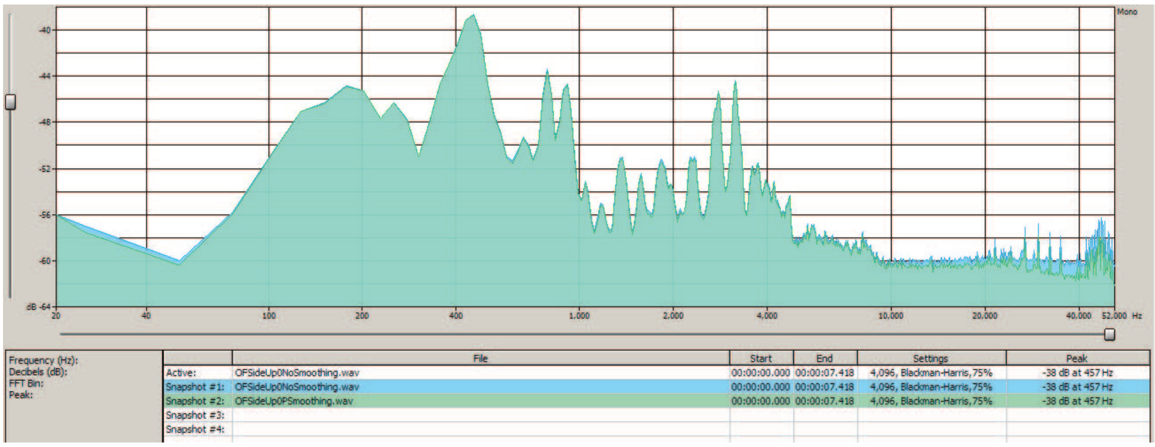
Without smoothing in blue and with partial smoothing in green

10.5.1 Nobody16



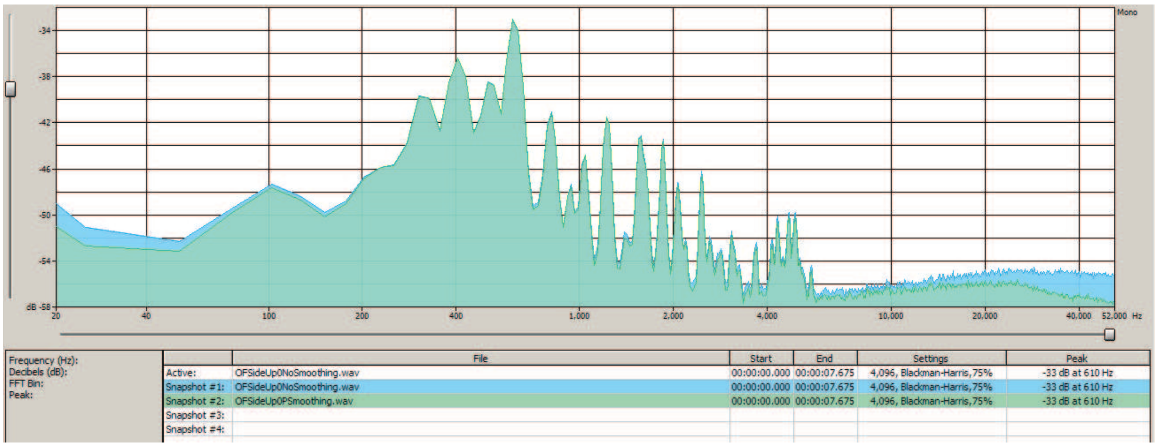
	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #2:	OFSideUp0PSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #3:					
Snapshot #4:					

10.5.2 SSB25



	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #2:	OFSideUp0PSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #3:					
Snapshot #4:					

10.5.3 DC1

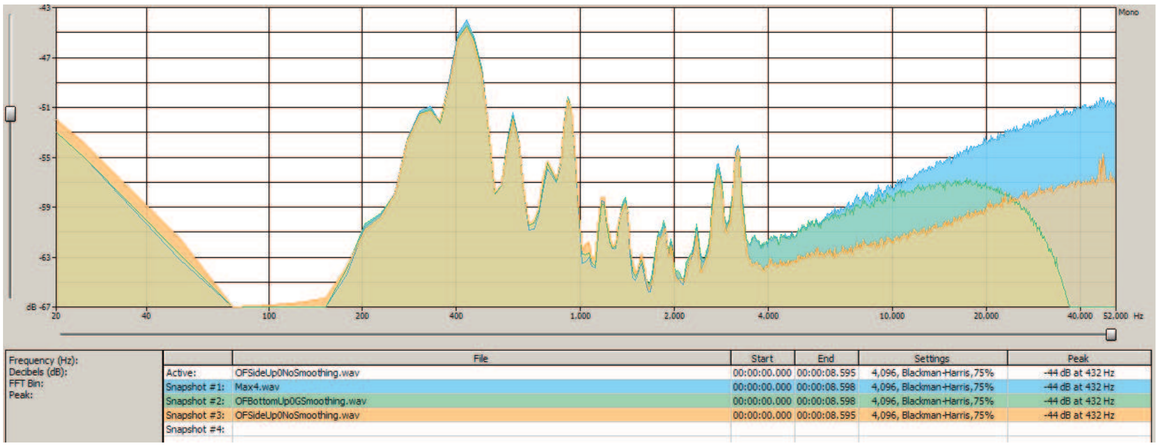


	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #1:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #2:	OFSideUp0PSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #3:					
Snapshot #4:					

10.6 Comparison with the best methods on music samples

Max derivative in blue, optical flow on normal images with Gaussian filter in green, optical flow on side-lightening images without smoothing in orange

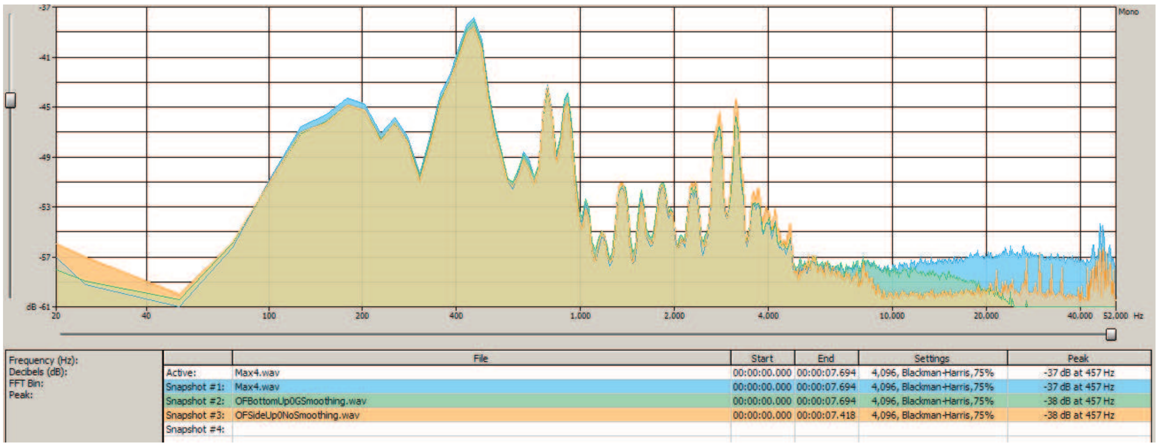
10.6.1 Nobody16



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:08.598	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:08.595	4,096, Blackman-Harris,75%	-44 dB at 432 Hz
Snapshot #4:					

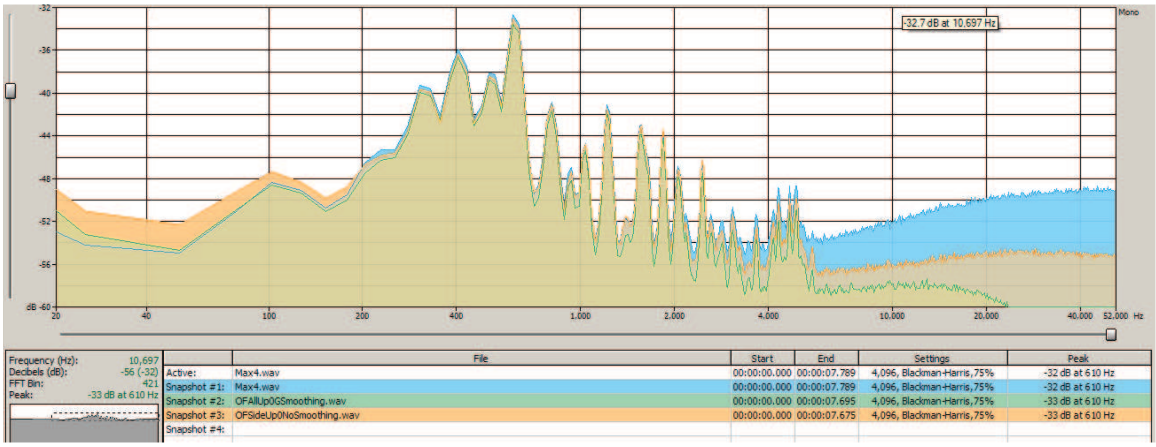
10.6.2 SSB25



Frequency (Hz):
Decibels (dB):
FFT Bin:
Peak:

	File	Start	End	Settings	Peak
Active:	Max4.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-37 dB at 457 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-37 dB at 457 Hz
Snapshot #2:	OFBottomUp0GSmoothing.wav	00:00:00.000	00:00:07.694	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.418	4,096, Blackman-Harris,75%	-38 dB at 457 Hz
Snapshot #4:					

10.6.3 DC1

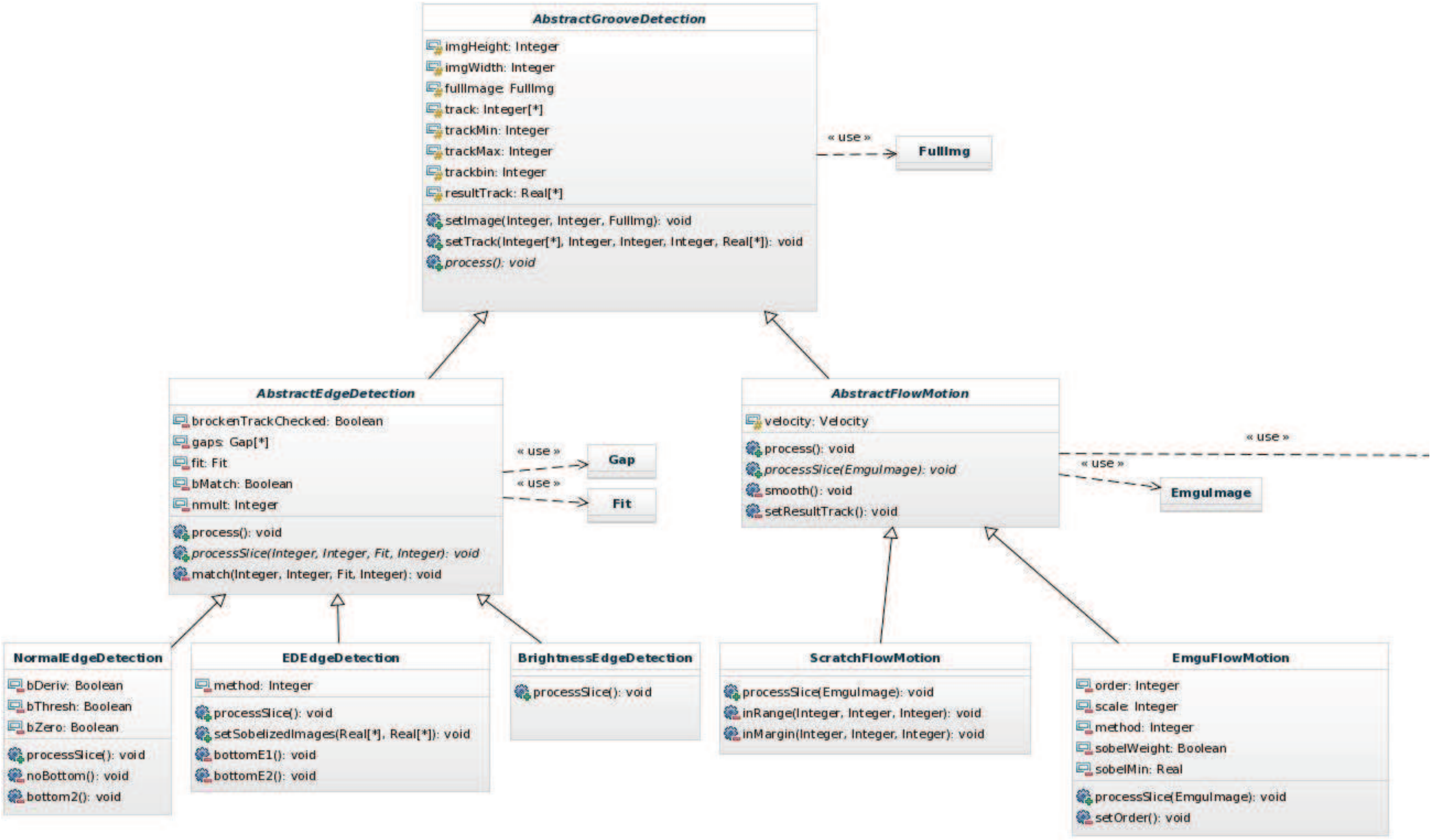


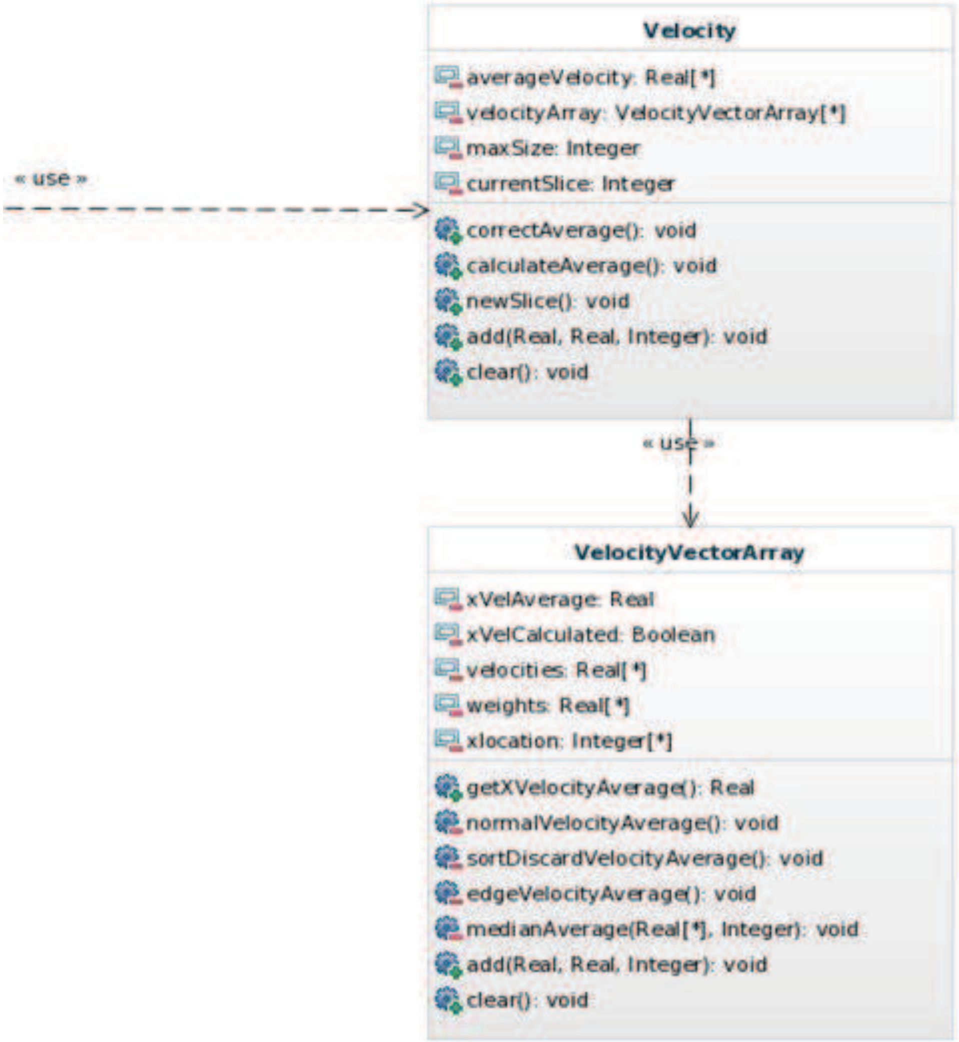
Frequency (Hz): 10,697
Decibels (dB): -56 (-32)
FFT Bin: 421
Peak: -33 dB at 610 Hz

	File	Start	End	Settings	Peak
Active:	Max4.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris,75%	-32 dB at 610 Hz
Snapshot #1:	Max4.wav	00:00:00.000	00:00:07.789	4,096, Blackman-Harris,75%	-32 dB at 610 Hz
Snapshot #2:	OFAllUp0GSmoothing.wav	00:00:00.000	00:00:07.695	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #3:	OFSideUp0NoSmoothing.wav	00:00:00.000	00:00:07.675	4,096, Blackman-Harris,75%	-33 dB at 610 Hz
Snapshot #4:					

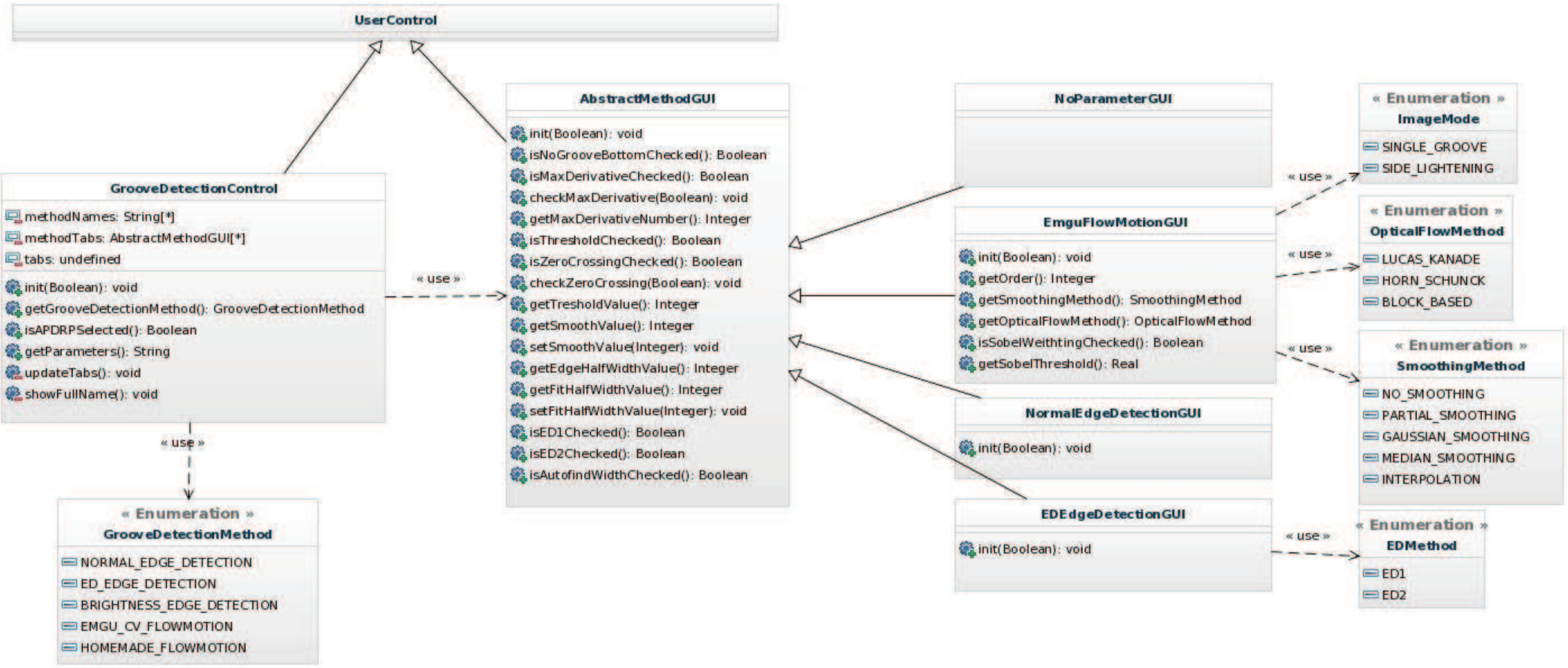
10.7 Class diagrams

10.7.1 Algorithmic part

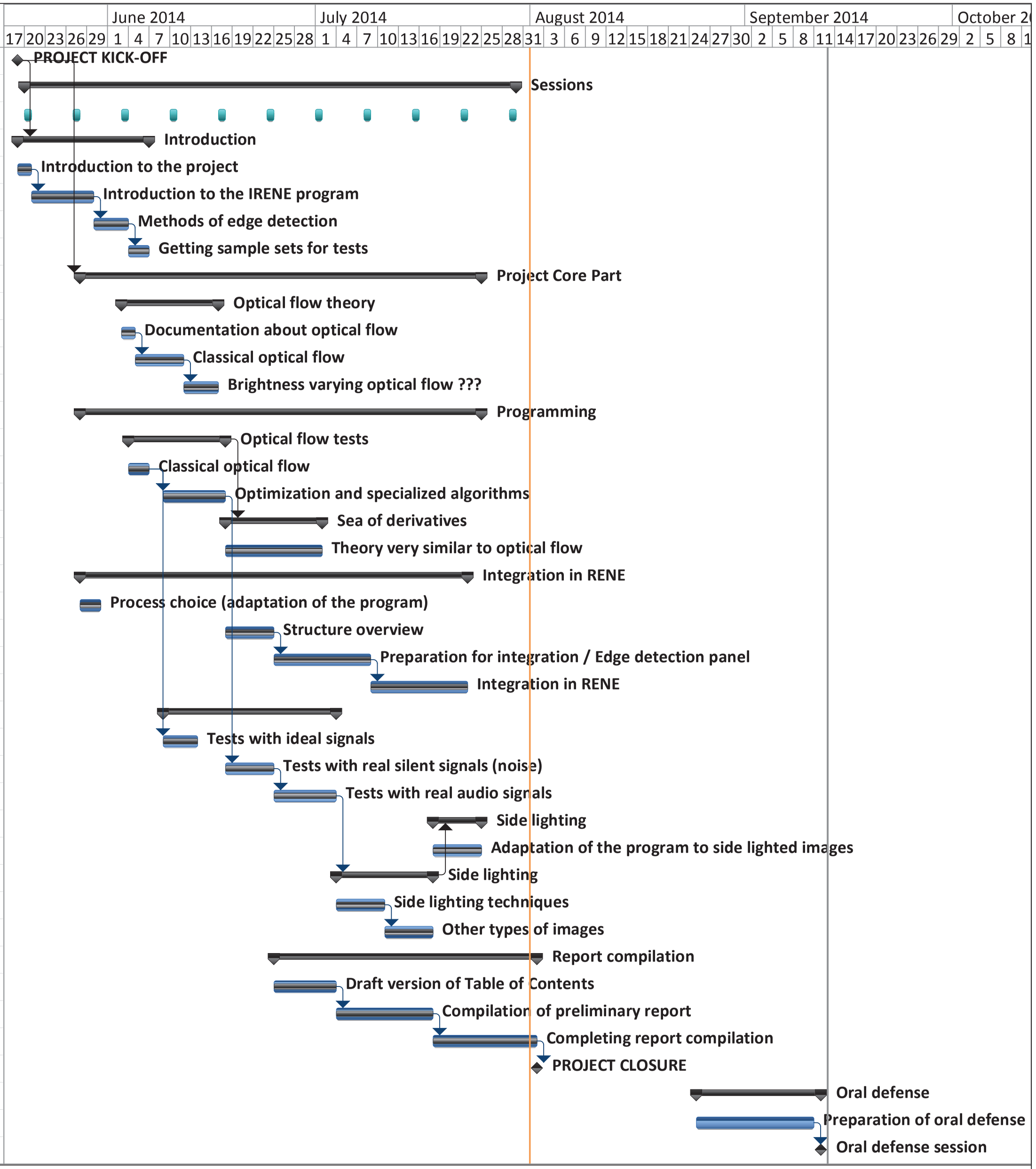



« use »
Velocity
averageVelocity: Real[*]
velocityArray: VelocityVectorArray[*]
maxSize: Integer
currentSlice: Integer
correctAverage(): void
calculateAverage(): void
newSlice(): void
add(Real, Real, Integer): void
clear(): void
« use »
VelocityVectorArray
xVelAverage: Real
xVelCalculated: Boolean
velocities: Real[*]
weights: Real[*]
xlocation: Integer[*]
getXVelocityAverage(): Real
normalVelocityAverage(): void
sortDiscardVelocityAverage(): void
edgeVelocityAverage(): void
medianAverage(Real[*], Integer): void
add(Real, Real, Integer): void
clear(): void

10.7.2 GUI


UserControl
GrooveDetectionControl
methodNames: String[*]
methodTabs: AbstractMethodGUI[*]
tabs: undefined
init(Boolean): void
getGrooveDetectionMethod(): GrooveDetectionMethod
isAPDRPSelected(): Boolean
getParameters(): String
updateTabs(): void
showFullName(): void
« use »
« Enumeration »
GrooveDetectionMethod
NORMAL_EDGE_DETECTION
ED_EDGE_DETECTION
BRIGHTNESS_EDGE_DETECTION
EMGU_CV_FLOWMOTION
HOMEMADE_FLOWMOTION
« use »
AbstractMethodGUI
init(Boolean): void
isNoGrooveBottomChecked(): Boolean
isMaxDerivativeChecked(): Boolean
checkMaxDerivative(Boolean): void
getMaxDerivativeNumber(): Integer
isThresholdChecked(): Boolean
isZeroCrossingChecked(): Boolean
checkZeroCrossing(Boolean): void
getTresholdValue(): Integer
getSmoothValue(): Integer
setSmoothValue(Integer): void
getEdgeHalfWidthValue(): Integer
getFitHalfWidthValue(): Integer
setFitHalfWidthValue(Integer): void
isED1Checked(): Boolean
isED2Checked(): Boolean
isAutofindWidthChecked(): Boolean
NoParameterGUI
EmguFlowMotionGUI
init(Boolean): void
getOrder(): Integer
getSmoothingMethod(): SmoothingMethod
getOpticalFlowMethod(): OpticalFlowMethod
isSobelWeithtingChecked(): Boolean
getSobelThreshold(): Real
NormalEdgeDetectionGUI
init(Boolean): void
EDEdgeDetectionGUI
init(Boolean): void
« use »
« Enumeration »
ImageMode
SINGLE_GROOVE
SIDE_LIGHTENING
« use »
« Enumeration »
OpticalFlowMethod
LUCAS_KANADE
HORN_SCHUNCK
BLOCK_BASED
« use »
« Enumeration »
SmoothingMethod
NO_SMOOTHING
PARTIAL_SMOOTHING
GAUSSIAN_SMOOTHING
MEDIAN_SMOOTHING
INTERPOLATION
« use »
« Enumeration »
EDMethod
ED1
ED2

ID	ℹ	Task Mode	Nom de la tâche	Duration	Start	Finish
1	✓		PROJECT KICK-OFF	0 days	Mon 5/19/14	Mon 5/19/14
2	✓		**Sessions**	**51 days**	Tue 5/20/14	**Tue 7/29/14**
3	↻		**Session**	**51 days**	**Tue 5/20/14**	**Tue 7/29/14**
15	✓		**Introduction**	**15 days**	Mon 5/19/14	**Fri 6/6/14**
16	✓		Introduction to the project	2 days	Mon 5/19/14	Tue 5/20/14
17	✓		Introduction to the IRENE program	7 days	Wed 5/21/14	Thu 5/29/14
18	✓		Methods of edge detection	3 days	Fri 5/30/14	Tue 6/3/14
19	✓		Getting sample sets for tests	3 days	Wed 6/4/14	Fri 6/6/14
20	✓		**Project Core Part**	**42 days**	Wed 5/28/14	**Thu 7/24/14**
21	✓		**Optical flow theory**	**10 days**	**Tue 6/3/14**	**Mon 6/16/14**
22	✓		Documentation about optical flow	2 days	Tue 6/3/14	Wed 6/4/14
23	✓		Classical optical flow	5 days	Thu 6/5/14	Wed 6/11/14
24	✓		Brightness varying optical flow ???	3 days	Thu 6/12/14	Mon 6/16/14
25	✓		**Programming**	**42 days**	**Wed 5/28/14**	**Thu 7/24/14**
26	✓		**Optical flow tests**	**10 days**	**Wed 6/4/14**	**Tue 6/17/14**
27	✓		Classical optical flow	3 days	Wed 6/4/14	Fri 6/6/14
28	✓		Optimization and specialized algorithms	7 days	Mon 6/9/14	Tue 6/17/14
29	✓		**Sea of derivatives**	**10 days**	**Wed 6/18/14**	**Tue 7/1/14**
30	✓		Theory very similar to optical flow	10 days	Wed 6/18/14	Tue 7/1/14
31	✓		**Integration in RENE**	**40 days**	**Wed 5/28/14**	**Tue 7/22/14**
32	✓		Process choice (adaptation of the program)	3 days	Wed 5/28/14	Fri 5/30/14
33	✓		Structure overview	5 days	Wed 6/18/14	Tue 6/24/14
34	✓		Preparation for integration / Edge detection panel	10 days	Wed 6/25/14	Tue 7/8/14
35	✓		Integration in RENE	10 days	Wed 7/9/14	Tue 7/22/14
36	✓		**Practical tests**	**19 days**	**Mon 6/9/14**	**Thu 7/3/14**
37	✓		Tests with ideal signals	5 days	Mon 6/9/14	Fri 6/13/14
38	✓		Tests with real silent signals (noise)	5 days	Wed 6/18/14	Tue 6/24/14
39	✓		Tests with real audio signals	7 days	Wed 6/25/14	Thu 7/3/14
40	✓		**Side lighting**	**5 days**	**Fri 7/18/14**	**Thu 7/24/14**
41	✓		Adaptation of the program to side lighted images	5 days	Fri 7/18/14	Thu 7/24/14
42	✓		**Side lighting**	**10 days**	**Fri 7/4/14**	**Thu 7/17/14**
43	✓		Side lighting techniques	5 days	Fri 7/4/14	Thu 7/10/14
44	✓		Other types of images	5 days	Fri 7/11/14	Thu 7/17/14
45	✓		**Report compilation**	**28 days**	Wed 6/25/14	**Fri 8/1/14**
46	✓		Draft version of Table of Contents	7 days	Wed 6/25/14	Thu 7/3/14
47	✓		Compilation of preliminary report	10 days	Fri 7/4/14	Thu 7/17/14
48	✓		Completing report compilation	11 days	Fri 7/18/14	Fri 8/1/14
49	▦		**PROJECT CLOSURE**	0 days	Fri 8/1/14	Fri 8/1/14
50			**Oral defense**	**14 days**	**Mon 8/25/14**	**Fri 9/12/14**
51	▦		Preparation of oral defense	13 days	Mon 8/25/14	Wed 9/10/14
52	▦		Oral defense session	0 days	Fri 9/12/14	Fri 9/12/14



Projet : Gant V2.0
Date : Fri 8/1/14

Tâche	▬	Tâches externes	▬	Tâche manuelle	▬	Fin uniquement	⊐
Fractionnement	┈	Jalons externes	◆	Durée uniquement	▬	Échéance	⇩
Jalon	◆	Tâche inactive	▭	Report récapitulatif manuel	▬	Avancement	▬
Récapitulative	▬	Jalon inactif	◇	Récapitulatif manuel	▬		
Récapitulatif du projet	▬	Récapitulatif inactif	▭	Début uniquement	⊏		